

AD-A164 842

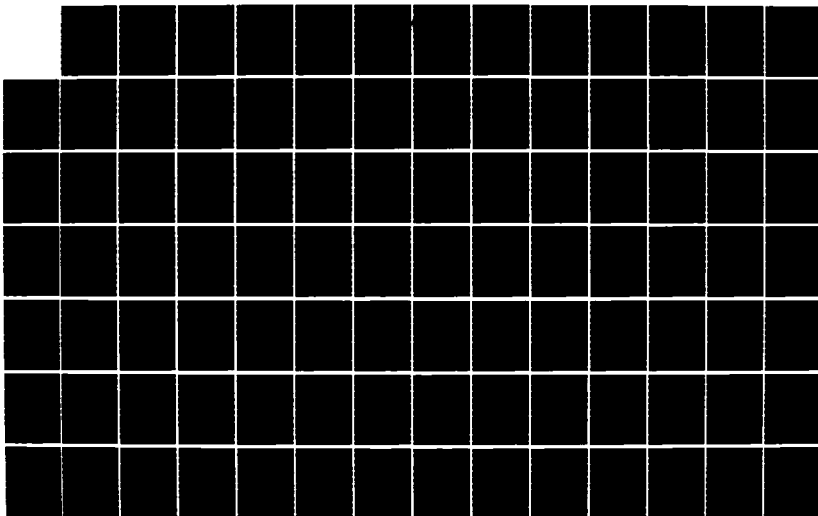
DEVELOPMENT AND IMPLEMENTATION OF THE X25 PROTOCOL FOR
THE UNIVERSAL NETW. (U) AIR FORCE INST OF TECH

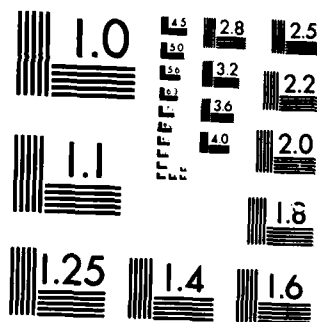
1/5

WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI... H W NEBER
DEC 85 AFIT/GE/ENG/85D-52-VOL-2 F/G 17/2

UNCLASSIFIED

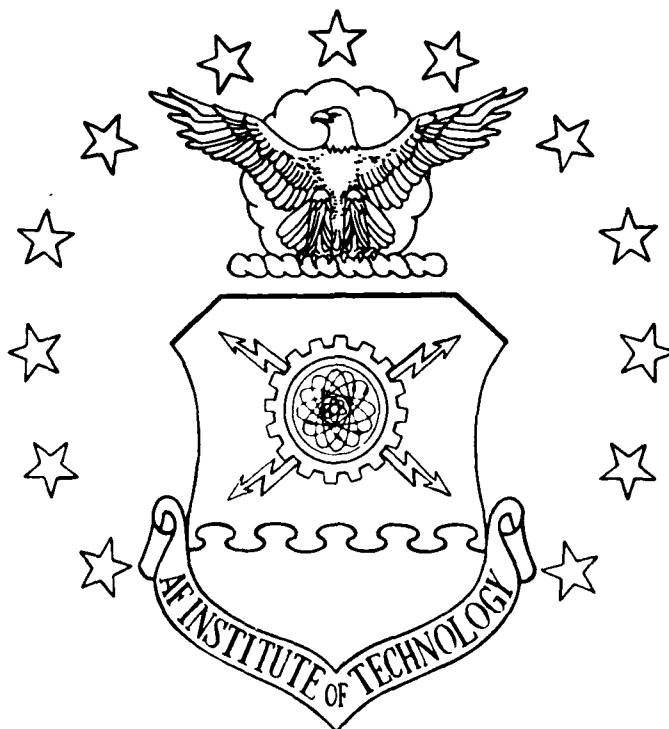
NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

AD-A164 042



DEVELOPMENT AND IMPLEMENTATION
OF THE
X.25 PROTOCOL
FOR THE
UNIVERSAL NETWORK INTERFACE DEVICE (UNID) II
VOL II OF II

THESIS

AFIT-TR-80-52

Mark W. Weber
Captain

USAF

FILE COPY

DTIC
ELECTE
FEB 15 1986

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

This document has been approved
for public release and its
distribution is unlimited.

86 2 12 027

①

DEVELOPMENT AND IMPLEMENTATION
OF THE
X.25 PROTOCOL
FOR THE
UNIVERSAL NETWORK INTERFACE DEVICE (UNID) II
VOL II OF II

THESIS

AFIT/GE/ENG/85D-52

Mark W. Weber
Captain

USAF

DTIC
ELECTE
FEB 13 1986
E

DISCLAIMER NOTICE

**THIS DOCUMENT IS BEST QUALITY
PRACTICABLE. THE COPY FURNISHED
TO DTIC CONTAINED A SIGNIFICANT
NUMBER OF PAGES WHICH DO NOT
REPRODUCE LEGIBLY.**

DEVELOPMENT AND IMPLEMENTATION
OF THE
X.25 PROTOCOL
FOR THE
UNIVERSAL NETWORK INTERFACE DEVICE (UNID) II
VOL II OF II

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
in Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Electrical Engineering

Mark W. Weber, BS EE
Captain, USAF

December 1985



Accession For	
NTIS GRA&I	<input checked="checked" type="checkbox"/>
DTIC TAB	<input checked="checked" type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By _____	
Distribution/ _____	
Availability Codes	
Dist	Avail and/or Special
A-1	23

APPENDIX J

ISIS HOST SOFTWARE

ISHOST.SRC may be linked with the I/O modules HOST2.SRC, or HOST3.SRC. The program is derived from Childress' host program CPMTMP.SRC (15). The software operates on the Intel 230 or Intel 210 under the ISIS operation system.

STY. E. LOCAL NETWORK TEST PROG FOR ISIS SYSTEM III. 15 AUG 1985')

```

*****
MAIN: DO;
*****
//** DATE: 15 AUG 85 //**
//** VERSION: 1.1 //**
//** TITLE: ISIS HOST SOFTWARE //**
//** FILENAME: ISHOST.SRC //**
//** OWNER: Capt Mark Weber //**
//** OPERATING SYSTEM: ISIS //**
//** LANGUAGE: INTEL PL/M 80 //**
//** USE: Link with the assemble driver HOST2.ASM or the PL/M driver //**
//** HOST3.SRC and the PLM80.LIB file and the SYSTEM.LIB files. //**
//** The submit file LNK4.CSD may be used to link and locate these //**
//** four files for use on the intel 230. The files must be located //**
//** at or above 04100 h to function properly. //**
//** CONTENTS: //**
//** DELAAY - time delay procedure //**
//** HEX$ASC - changes hex numbers to ascii characters //**
//** ASC$HEX - changes ascii characters to hex numbers //**
//** VALID$HEX - checks for true hex number //**
//** INIT - initialization routine for variable and channel //**
//** ERR$CHK - checks status of ISIS called routines //**
//** SNDSEQ - writes to the console //**
//** READ$CON - reads the system console //**
//** RCV$1 - sets channel one up as a monitor for the SBC 54 //**
//** TRANS$1 - sets channel one up to transmit data to the //**
//** channel one monitor //**
//** LOSTAB$H$SKP - house keeps the specified table //**
//** SRVC$TAB$H$SKP - services table when packet is removed //**
//** CHK$RX$TA - checks the netos receive acknowledge //**
//** CHK$TX$TR - checks for the transmit request character //**
//** READ$TX$TAB - reads the transmit tables //**
//** READ$RX$TAB - reads the receive tables //**
//** LOAD - places a message in the TX01TB table //**
//** READ$LINE - Reads a line of data from the host system //**
//** LOOP2 - loops data back to the system console through //**
//** channel one //**
//** HISTORY: //**
//** 1.1 - Capt Mark Weber - abopted to ISI //**
//** systems with X.25 protocol specifications //**
//** 1.0 - Capt Creed T. Childress - origanal //**
//** CPM version //**
//** ***** EXTERNAL PROCEDURE DECLARATIONS ***** //**
//**

```

J-3

```

/* FOLLOWING ARE NETWORK DEFINED VARIABLES */
/* NOTES:1. THIS$UNID$NBR MUST REFLECT WHICH UNID THIS IS.
2. THIS$COUNTRY$CODE MUST REFLECT THE AREA TO WHICH
THIS UNID IS LOCATED.
3. MAX$COUNTRY$CODE WILL INDICATE WHICH COUNTRY CODES
ARE CURRENTLY OPERATIONAL. CC=0000 IS RESERVED FOR
THE DELNET MONITOR.
4. MAX$NETWORK$CODE WILL INDICATE HOW MANY UNIDS ARE
CURRENTLY OPERATIONAL WITHIN A PARTICULAR COUNTRY.
5. FOR DETAILED INFORMATION ON THE ABOVE REFER TO
PHISTER'S THESIS, APPENDIX D. */
THIS$UNID$NBR LITERALLY '2'; /* UNIQUE ADDRESS FOR THIS UNID */
THIS$COUNTRY$CODE LITERALLY '9'; /* CC WHERE THIS UNID RESIDES */
MAX$COUNTRY$CODE LITERALLY '9'; /* INDICATES COUNTRY$CODES IN USE */
MAX$NETWORK$CODE LITERALLY '3'; /* INDICATES UNIDS OPERATIONAL IN NET */

```

```

/*****
/* ADDITIONAL GENERAL DECLARES NEEDED FOR THIS PROGRAM
*****/

```

```

DECLARE

```

```

CHAN$PTR ADDRESS.

```

```

RXSTA$TRIES BYTE,
TEMP1(*) BYTE INITIAL('XEMPTY EMPTYV0XX'),
TEMP2(*) BYTE INITIAL('XEMPTY EMPTYV0XX'),
(TRTA, TXTR, TXTA, RXTR, RXTA, CHAR) BYTE,
FOREVER BYTE,
TRANSS$1$RDV BYTE,
BYTES$SENT$1 BYTE,
BYTES$RECV$1 BYTE,
MSGNUM BYTE,

```

```

/*****
/* DATA TABLES USED IN THIS PROGRAM
*****/

```

```

RX01TB (DATA$TABLE$SIZE) BYTE,
RX01NS ADDRESS,
RX01NE ADDRESS,
RX01SZ ADDRESS,

```

```

TX01TB (DATA$TABLE$SIZE) BYTE,
TX01NS ADDRESS,
TX01NE ADDRESS,
TX01SZ ADDRESS,

```

```

DESTINAT
DESTIN
SOURCE$ADDR

```

```

/* MISCELLANEOUS DECLARATIONS */

DECLARE
  TR      LITERALLY '42H'; /* TR, TA FOLLOWS NETOS */
  TA      LITERALLY '41H'; /* CONVENTION */
  BUSY    LITERALLY 'OFFH';
  TRUE    LITERALLY 'OFFH';
  FALSE   LITERALLY '00H';
  NMBR$MSK LITERALLY '07H';
  CR      LITERALLY '0DH';
  LF      LITERALLY '0AH';
  SOURCE  LITERALLY '12';
  DESTIN  LITERALLY '16';
  ESC     LITERALLY '1BH';
  RCV$STATE LITERALLY '00010110B'; /* MODE INSTRUCTION FOR */
                                     /* 8251 USART RCV INT */
  TRANS$STATE LITERALLY '00110111B'; /* MODE INSTRUCTION FOR */
                                     /* 8251 RCV & XMIT INT */

  STARTUP$HDR(*) BYTE DATA(CR,LF,
    UNID II #2 LOCAL OS',CR,LF,
    SUPPORTING S/W ON ISIS',CR,LF,
    VERS 1.1, 25 SEP 85',CR,LF,
    EXECUTING ',CR,LF);

DECLARE
  INIT$MSG(*) BYTE DATA(CR,LF,
    CONFIGURED for the ISIS SYSTEM II at 1.18 Mhz',CR,LF,
    timing to the I/O port.',CR,LF,LF,
    Operating with the UNID II ISBC TCP/IP protocol',CR,LF,
    to simulate a host system',CR,LF);

/* THE FOLLOWING TEST POINTS ARE USED TO FOLLOW THE DATA WITHIN THE UNID */

DECLARE TP$50(*) BYTE DATA(CR,LF,
  Channel Number = ');
DECLARE TP$51A(*) BYTE DATA(CR,LF,
  Destination Network = ');
DECLARE TP$52A(*) BYTE DATA(CR,LF,
  Destination Host = ');
DECLARE TP$54(*) BYTE DATA(CR,LF,
  Loaded test datagram in TX01TB');
DECLARE TP$55A(*) BYTE DATA(CR,LF,
  Reading test datagram from TX01TB');
DECLARE TP$56A(*) BYTE DATA(CR,LF,
  Reading test datagram from RX01TB');
DECLARE TP$57(*) BYTE DATA(CR,LF,
  Sent TR');
DECLARE TP$58(*) BYTE DATA(CR,LF,
  Sent TA');
DECLARE TP$59(*) BYTE DATA(CR,LF,

```

SOURCE CODE FOR ISHOST.SRC, 15 AUG 85

```
'TP$59      Received TR');
DECLARE TP$60(*) BYTE DATA(CR,LF,
'Received TA');
DECLARE MSG1(*)   BYTE DATA(CR,LF,
'Do you want to load the test message (Y/N) [N] ? ');
DECLARE MSG1A(*) BYTE DATA(CR,LF,
'Do you want TR/TA handshake (Y/N) [N] ? ');
DECLARE MSG2(*)  BYTE DATA(CR,LF,
'Do you want to stop the test (Y/N) [N] ? ');
DECLARE MSG3(*)  BYTE DATA(CR,LF,
'Load into which host channel (1,2,3,4)? ');
DECLARE MSG4(*)  BYTE DATA(CR,LF,
'How many datagrams (0 - 9)? ');
DECLARE MSG5(*)  BYTE DATA(CR,LF,
'Destination network code (1,2,3) = ');
DECLARE MSG6(*)  BYTE DATA(CR,LF,
'Destination host code (0 - FFH) = ');
DECLARE MSG7(*)  BYTE DATA(CR,LF,
'Finished simulation, rebooting to ISIS',CR,LF);
DECLARE MSG8(*)  BYTE DATA(CR,LF,
'Timeout on receive datagram from UNID');
DECLARE MSG9(*)  BYTE DATA(CR,LF,
'Timeout on receive TA from UNID');
DECLARE MSG10(*) BYTE DATA(CR,LF,
'Timeout on receive TR from UNID');
```

```
/****** 25 SEP 84 *****
* DATE: 25 SEP 84
* VERSION: 1.0
* NAME: DELAY
* MODULE NUMBER: Not yet implemented
* FUNCTION: The purpose of this procedure is to add a
            delay to parts of the initialization procedure that
            are time dependent. Delay is approximately in
            milliseconds.
* INPUTS: MILLISEC (approximate)
* OUTPUTS: None
* GLOBAL VARIABLES USED: None
* GLOBAL VARIABLES CHANGED: None
* MODULES CALLED: TIME (PLM80.LIB)
* CALLING MODULES: None yet
* AUTHOR: Capt C. T. Childress
* HISTORY:
            1.0 - Capt C. T. Childress -25 SEP 85 configured for
            CP/M
*****/
DELAY: PROCEDURE(MILLISEC);
DECLARE (I, MILLISEC) BYTE;
```


SOURCE CODE FOR ISHOST.SRC, 15 AUG 85

```
DO I=1 TO MILLISEC;
  CALL TIME(25); /* TIME IS A BUILT IN FUNCTION OF PLM80 WHICH CAUSES */
                /* A DELAY BASED ON THE NUMBER IN PARENS */
END;
```

```
END DELAY;
```

```
/* *****
PROCEDURES TO CONVERT HEX TO ASCII FOR USE WITH DISPLAYING A FRAME
***** */
/* *****
DATE: 25 SEP 84
VERSION: 1.0
NAME: HEX$ASC
MODULE NUMBER: 4.1
FUNCTION: Converts a HEX number into its ascii characters
          for printing to console
INPUTS: VAL - a hex number
OUTPUTS: HEX$ASC
GLOBAL VARIABLES USED: ASCII - array of legal hex characters
                      I - position in string
                      TEMPI - ASCII output array
                      None
GLOBAL VARIABLES CHANGED: None
MODULES CALLED:
CALLING MODULES: SNDSEQ, READ$TX$TAB
AUTHOR: Capt C.T. Childress
HISTORY: 1.0 - Capt C. T. Childress -25 SEP 84
***** */
HEX$ASC: PROCEDURE(VAL, I);
DECLARE (VAL, I) BYTE;
```

```
  TEMPI(I) = ASCII(SHR(VAL,4) AND 0FH);
  TEMPI(I+1) = ASCII(VAL AND 0FH);
```

```
END HEX$ASC;
```

```
/* *****
DATE: 25 SEP 84
VERSION: 1.0
NAME: ASC$HEX
MODULE NUMBER: 3.3
FUNCTION: Converts ASCII characters to their HEX equivalent number.
          C - the character
INPUTS: C - the character
OUTPUTS: ASC$HEX
GLOBAL VARIABLES USED: None
GLOBAL VARIABLES CHANGED: None
MODULES CALLED: None
CALLING MODULES: READLINE
AUTHOR: Capt C.T. Childress
HISTORY:
***** */
```

SOURCE CODE FOR ISHOST.SRC, 15 AUG 85

```

*
* 1.0 - Capt C. T. Childress -25 SEP 84
* *****
ASC$HEX: PROCEDURE(C) BYTE;
DECLARE C BYTE;

```

```

IF (C >= '0' AND C <= '9') THEN
  RETURN (C-30H);
ELSE
  IF (C >= 'A' AND C <= 'F') THEN
    RETURN (C-37H);
  ELSE
    IF (C >= 'a' AND C <= 'f') THEN
      RETURN (C-57H);

```

RETURN C;

END ASC\$HEX;

```

/*****
* DATE: 25 SEP 84
* VERSION: 1.0
* NAME: VALID$HEX
* MODULE NUMBER: Not yet used
* FUNCTION: Validates a char to be changed to hex
* INPUTS: H - char
* OUTPUTS: VALIDHEX - boolean
* GLOBAL VARIABLES USED: ASCII - array of valid ascii char
* GLOBAL VARIABLES CHANGED: None
* MODULES CALLED: None
* CALLING MODULES: None yet
* AUTHOR: Capt C.T. Childress
* HISTORY:
* 1.0 - Capt C. T. Childress -25 SEP 84
* *****
VALID$HEX: PROCEDURE(H) BYTE;
DECLARE (H,I) BYTE;

```

```

DO I = 0 TO LAST(ASCII);
  IF H = ASCII(I) THEN
    RETURN TRUE;

```

```

END;
RETURN FALSE;

```

END VALID\$HEX;

```

/*****
* DATE: 24 SEP 84
* VERSION: 1.0
*

```

SOURCE CODE FOR ISHOST.SRC, 15 AUG 85

```

* NAME: INIT
* MODULE NUMBER: 1.0
* FUNCTION: Initializes the software including all global
*          variables, tables, pointers and the communications
*          channel. This procedure writes the control words to
*          the PIC, PIT, PPI, and USARTs.
* INPUTS: None
* OUTPUTS: None
* GLOBAL VARIABLES USED: MSGNUM, BYTESSENT$1, BYTES$RECV$1, CHAN$NUM
*                       DEST$NET$CODE, BUFFER, CHAR, TRANS$1$RDY, RXTA$TRIES
*                       TRTA, TXTR, RXTR, RXTA, RX01NS, RX01NE, RX01SZ, TX01NS,
*                       TX01NE, TX01SZ
* GLOBAL VARIABLES CHANGED: same as above
* MODULES CALLED: SINIT, SCLRCM
* CALLING MODULES: main
* AUTHOR: Capt C.T. Childress
* HISTORY:
*          1.0 - Capt C. T. Childress -25 SEP 84
*****
INIT: PROCEDURE;
*****
    MSGNUM = 0;
    BYTES$SENT$1 = 0;
    BYTES$RECV$1 = 0;
    CHAN$NUM = 0;
    DEST$NET$CODE = 0;
    BUFFER(0) = 120;
    CHAR = ' ';
    TRANS$1$RDY = FALSE;
    RXTA$TRIES = 0;
    TRTA, TXTR, TXTA, RXTR, RXTA = FALSE;

/*****
/*
/*      THIS PORTION SETS THE LOCAL AND NETWORK TABLES TO THIER
/*      INITIAL VALUES PRIOR TO PROCESSING ANY MESSAGES.
/*
/*
*****/
*****
    RX01NS = 0;
    RX01NE = 0;
    RX01SZ = DATA$TABLE$SIZE;

    TX01NS = 0;
    TX01NE = 0;
    TX01SZ = DATA$TABLE$SIZE;

    CALL SINIT;
    CALL SCLRCM;

```

END INIT;

```

/*****
* DATE: 15 AUG 85
* VERSION: 1.1
* NAME: ERR$CHK
* MODULE NUMBER: 2.1
* FUNCTION: Checks the status of ISIS call procedures and
* for an orderly exit incase of error
* INPUTS: None
* OUTPUTS: None
* GLOBAL VARIABLES USED: STATUS
* GLOBAL VARIABLES CHANGED: None
* MODULES CALLED: ERROR, EXIT
* CALLING MODULES: SNDSEQ, READ$CON
* AUTHOR: Capt Mark Weber
* HISTORY:
* 1.1 - Capt Mark Weber -15 AUG 85 changed from CPM exit to
* ISIS exit procedures
* 1.0 - Capt C. T. Childress 25 SEP 84
*****/

```

```

ERR$CHK: PROCEDURE;
  IF STATUS <> 0 THEN
    DO;
      CALL ERROR(STATUS);
    CALL EXIT;
  END;

```

END ERR\$CHK;

```

/*****
* DATE: 15 AUG 85
* VERSION: 1.1
* NAME: SNDSEQ
* MODULE NUMBER: 2.0
* FUNCTION: Sends a message to the system console
* INPUTS: MSG - the message address
* TOTAL - the count of the char in the buffer
* OUTPUTS: None (Text to console output)
* GLOBAL VARIABLES USED: STATUS
* GLOBAL VARIABLES CHANGED: STATUS
* MODULES CALLED: ERR$CHK, WRITE (ISIS call)
* CALLING MODULES: main, READ$LINE, LOAD, READ$RX$TAB,
* READ$TX$TAB, RCV$I
* AUTHOR: Capt Mark Weber
* HISTORY:
*****/

```

*
* 1.1 - Capt Mark Weber -15 AUG 85 -configured for ISIS
*
* 1.0 - Capt C. T. Childress -25 SEP 84 configured for
* CP/M

SNDSEQ: PROCEDURE(MSG, TOTAL) REENTRANT;
DECLARE MSG ADDRESS,
TOTAL ADDRESS;

```
CALL WRITE(0, MSG, TOTAL, .STATUS);
CALL ERR$CHK;
```

END SNDSEQ:

```

/*****
** DATE: 15 AUG 85
** VERSION: 2.0
** NAME: READ$CON
** MODULE NUMBER: 3.2
** FUNCTION: Reads a character from the system console
** INPUTS: None
** OUTPUTS: None
** GLOBAL VARIABLES USED: BUFFER, ACT$COUNT, STATUS
** GLOBAL VARIABLES CHANGED: BUFFER, ACT$COUNT, STATUS
** MODULES CALLED: ERR$CHK, READ (ISIS call), WRITE(ISIS)
** CALLING MODULES: READ$LINE
** AUTHOR: Capt Mark Weber
** HISTORY:
** 1.1 - Capt Mark Weber -15 AUG 85 -configured for ISIS
** 1.0 - Capt C. T. Childress -25 SEP 85 configured for
** Cp/M
*****/

```

READ\$CON: PROCEDURE;

```
CALL READ(1, .BUFFER, 128, .ACT$COUNT, .STATUS);
      CALL ERR$CHK;
```

```
CALL WRITE(0, .BUFFER, ACT$COUNT, .STATUS);
CALL ERR$CHK;
```

END READ\$CON;

```

/*****
*   DATE:      24 SEP 84
*   VERSION:    1.0
*   NAME:       RCV$1
*   MODULE NUMBER: 7.2
*   FUNCTION:    Receives data from channel one of SBC 544.
*               This procedure takes a character at a time from the
*               receive port and puts it in the receive channel buffer
*****/

```

SOURCE CODE FOR ISHOST.SRC, 15 AUG 85

```

*      The routine is interrupt driven and operates after
*      initialization. Operates off of USART 0
*
*      INPUTS:      None
*      OUTPUTS:     None
*      GLOBAL VARIABLES USED:
*          RXTR, TXTR, TXTR, RX01INE, RX01SZ
*          BYTES$RCV$1, RX01TB
*          same as above
*      GLOBAL VARIABLES CHANGED:
*          SCMCHK, SNDSEQ
*      MODULES CALLED:
*          LOOP2
*      CALLING MODULES:
*          AUTHOR: Capt Mark Weber
*      HISTORY:
*      HISTORY:
*      1.0 - Capt C. T. Childress -25 SEP 85 configured for
*            CP/M
*
*****
/* USE THIS CODE FOR SELF TEST; CHAR BY CHAR RECEIVE */
/*
RVC$1: PROCEDURE;

  RX01TB(RX01INE) = SCMIN;
  RX01INE = RX01INE + 1;
  IF(RX01INE >= RX01SZ) THEN
    RX01INE = 0;

END RVC$1;

/*
/* USE THE FOLLOWING FOR OPERATIONAL 544 S/W TEST */
/*
/* DATAGRAM BY DATAGRAM RECEIVE */
RVC$1: PROCEDURE;
  DECLARE COUNT ADDRESS;

  COUNT = 30000;
  DO WHILE ((SCMCHK = 0) AND (COUNT <> 0));
    COUNT = COUNT - 1;
  END;
  IF COUNT = 0 THEN
    DO;
      CALL SNDSEQ(.MSG8, SIZE(MSG8));
      RXTR = FALSE;
      TXTR = FALSE;
    END;
  ELSE
    IF ((NOT TRTA) OR (RXTR AND TXTR AND
      (NOT TXTR) AND (NOT RXTR))) THEN
      DO;
        DO WHILE (BYTES$RCV$1 < DATA$GRAM$SIZE);
          RX01TB(RX01INE) = SCMIN;
          RX01INE = RX01INE + 1;
          BYTES$RCV$1 = BYTES$RCV$1 + 1;

```

```

END;
BYTE$RECV$1 = 0;
IF (RX01NE >= RX01SZ) THEN
  RX01NE = 0;
  RXTR = FALSE;
  TXTA = FALSE;
END;

```

```

END RCV$1;

```

```

/*****
* DATE: 25 SEP 84
* VERSION: 1.0
* NAME: TRANS$1
* MODULE NUMBER: 7.1
* FUNCTION: This procedure sends a datagram out local
*           channel 1. Each byte of data generates an interrupt
*           on USART 0. When the message is complete the transmit
*           interrupt is disabled and TRANS$I RDT is set true.
* INPUTS: None
* OUTPUTS: None
* GLOBAL VARIABLES USED: BYTE$SENT$1, DATA$GRAM$SIZE, TX01TB
*                       TX01NS$I$RDV, TX01SZ, TXTR, RXTA
* GLOBAL VARIABLES CHANGED: same as above
* MODULES CALLED: SCMOU
* CALLING MODULES: LOOP2
* AUTHOR: Capt Mark Weber
* HISTORY:
*           1.0 - Capt C. T. Childress -25 SEP 85 configured for
*           CP/M
*****
/* USE THIS CODE FOR SELF TEST; CHAR BY CHAR TRANSMIT */
TRANS$1: PROCEDURE;

```

```

IF (BYTE$SENT$1 < DATA$GRAM$SIZE) THEN
DO;
  CALL SCMOU(TX01TB(TX01NS));
  BYTE$SENT$1 = BYTE$SENT$1 + 1;
  TX01NS = TX01NS + 1;
IF BYTE$SENT$1 >= DATA$GRAM$SIZE THEN
DO;
  BYTE$SENT$1 = 0;
  IF TX01NS >= TX01SZ THEN
    TX01NS = 0;
  TRANS$I$RDV = FALSE;
END;

```

```

END;

END TRANS$1;
/*
/* USE THE FOLLOWING FOR OPERATIONAL 544 S/W TEST */
/* DATAGRAM BY DATAGRAM TRANSMIT */

TRANS$1: PROCEDURE;

DO;
DO WHILE (BYTESSENT$1 < DATA$GRAM$SIZE);
CALL SCOUT(TX01TB(TX01NS));
BYTESSENT$1 = BYTESSENT$1 + 1;
TX01NS = TX01NS + 1;
END;
BYTESSENT$1 = 0;
IF TX01NS >= TX01SZ THEN
TX01NS = 0;
TRANS$1$RDV = FALSE;
TXTR = FALSE;
RXTA = FALSE;
END;

END TRANS$1;

/*****
* DATE: 25 SEP 84
* VERSION: 1.0
* NAME: LD$TAB$H$SKP
* MODULE NUMBER: 3.5
* FUNCTION: This procedure housekeeps a specified buffer
* after loading the user data from the host. The pro-
* cedure determines which table to process, and advances
* the NEX$EMPTY$BYTE address by one PACKET$SIZE, and if
* necessary wraps the data around to the beginning of
* the queue.
* INPUTS: TABLE - address of the table to process
* OUTPUTS: None
* GLOBAL VARIABLES USED: TX01NE, DATAGRAM$SIZE, TX01SZ
* GLOBAL VARIABLES CHANGED: TX01NE
* MODULES CALLED: NONE
* CALLING MODULES: READ$LINE
* AUTHOR: Capt Mark Weber
* HISTORY: 1.0 - Capt C. T. Childress -25 SEP 85 configured for
* CP/M
*****/
LD$TAB$H$SKP: PROCEDURE(TABLE);
DECLARE TABLE ADDRESS;

```


SOURCE CODE FOR ISHOST.SRC, 15 AUG 85

```

TX01NE = TX01NE + DATA$GRAM$SIZE;
IF TX01NE >= TX01SZ THEN
  TX01NE = 0;

```

```

END LD$TAB$H$SKP;

```

```

/***** 25 SEP 84 *****/
* DATE: 25 SEP 84
* VERSION: 1.0
* NAME: SRCV$TAB$H$SKP
* MODULE NUMBER: 4.3
* FUNCTION: This procedure housekeeps a specified table
* after servicing removing a packet. The procedure
* determines which table to process, advances the next
* byte to service pointer, and performs any necessary
* wrap around requirements
* INPUTS: TAB - pointer to table to process
* OUTPUTS: None
* GLOBAL VARIABLES USED: RX01NS, RX01SZ, TX01NS, TX01SZ,
* DATA$GRAM$SIZE
* GLOBAL VARIABLES CHANGED: RX01NS, RX01SZ, TX01NS, TX01SZ
* MODULES CALLED: None
* CALLING MODULES: READ$RX$TAB
* AUTHOR: Capt Mark Weber
* HISTORY:
* 1.0 - Capt C. T. Childress -25 SEP 85 configured for
* Cp/M
*****/
SRVC$TAB$H$SKP: PROCEDURE(TAB) ;
  DECLARE TAB ADDRESS;

```

```

DO CASE TAB;
;

```

```

DO;
  RX01NS = RX01NS + DATA$GRAM$SIZE;
  IF RX01NS >= RX01SZ THEN
    RX01NS = 0;
  END;

```

```

DO;
  TX01NS = TX01NS + DATA$GRAM$SIZE;
  IF TX01NS >= TX01SZ THEN
    TX01NS = 0;
  END;

```

```

END;

```

```

END SRVC$TAB$H$SKP;

```

```

/*****
* DATE: 15 AUG 85
* VERSION: 1.0
* NAME: CHK$RXTA
* MODULE NUMBER: 9.0
* FUNCTION: Checks for RXTA from UNID
* INPUTS: None
* OUTPUTS: None
* GLOBAL VARIABLES USED: TXTR,RXTA,RXTR,TXTA,RXTA$TRIES
* GLOBAL VARIABLES CHANGED: same as above
* MODULES CALLED: SCMCHK, SNDSEQ, SCMIN, TRANS$1
* CALLING MODULES: none yet
* AUTHOR: Capt C.T. Childress
* HISTORY:
* 1.0 - Capt C. T. Childress
*****/
CHK$RXTA: PROCEDURE;
  DECLARE COUNT ADDRESS;

  IF (TXTR AND (NOT RXTA) AND (NOT RXTR) AND (NOT TXTA)) THEN
    COUNT = 30000;
    DO WHILE ((SCMCHK = 0) AND (COUNT <> 0));
      COUNT = COUNT - 1;
    END;
    IF COUNT = 0 THEN
      DO;
        CALL SNDSEQ(.MSG9, SIZE(MSG9));
        TXTR = FALSE;
      END;
    ELSE
      DO;
        CHAR = SCMIN;
        IF CHAR = TA THEN
          DO;
            CALL SNDSEQ(.TP$60, SIZE(TP$60));
            RXTA = TRUE;
            RXTA$TRIES = 0;
            CALL TRANS$1;
          END;
        END;
      END;
  END;

END CHK$RXTA;

/*****
* DATE: 25 SEP 84
* VERSION: 1.0
* NAME: CHK$RXTR
* MODULE NUMBER: 10.0
* FUNCTION: Checks RXTR from the channel
* INPUTS: None
*****/

```

SOURCE CODE FOR ISHOST.SRC, 15 AUG 85

```

*   OUTPUTS:
*   GLOBAL VARIABLES USED:      RXTR, TXTA, TXTR, RXTA
*   GLOBAL VARIABLES CHANGED:  same as above
*   MODULES CALLED:             SCMCHK, SNDSEQ, SCMIN, SCMOUT, RCV$1
*   CALLING MODULES:            None yet
*   AUTHOR:  Capt C.T. Chrildress
*   HISTORY:
*   1.0 - Capt C. T. Chrildress
*   *****/
CHK$RXTR: PROCEDURE;
  DECLARE COUNT ADDRESS;

  IF ((NOT RXTR) AND (NOT TXTA) AND (NOT TXTR) AND (NOT RXTA)) THEN
    COUNT = 30000;
    DO WHILE ((SCMCHK = 0) AND (COUNT <> 0));
      COUNT = COUNT - 1;
    END;
    IF COUNT = 0 THEN
      CALL SNDSEQ(.MSG10, SIZE(MSG10));
    ELSE
      DO;
        CHAR = SCMIN;
        IF CHAR = 'R' THEN
          DO;
            CALL SNDSEQ(.TP$59, SIZE(TP$59));
            CALL SNDSEQ(.TP$58, SIZE(TP$58));
            RXTR = TRUE;
            TXTA = TRUE;
            CALL SCMOUT(TA);
            CALL RCV$1;
          END;
        END;
      END;
    END;
  END;

END CHK$RXTR;

```

```

/*****
*   DATE:      25 SEP 84
*   VERSION:   1.0
*   NAME:      READ$TX$TAB;
*   MODULE NUMBER: 5.0
*   FUNCTION:   Reads the local transmit table
*   INPUTS:     None
*   OUTPUTS:    None
*   GLOBAL VARIABLES USED:
*   RXTASTRIES, TXTR, RXTA, TXO1NE, TXO1SZ
*   DATA$GRAM$SIZE, MAX$RXTASTRIES
*   GLOBAL VARIABLES CHANGED:
*   RXTASTRIES, TXTR, TXO1NE, TXO1SZ, RXTA
*   MODULES CALLED:
*   SRVC$TAB$H$SKP, HEX$ASC, SNDSEQ, SCMOUT
*   CALLING MODULES:
*   main
*   AUTHOR:  Capt C.T. Chrildress
*

```

```

* HISTORY:
*          1.0 - Capt C. T. Childress
*          *****
READ$TXTAB:  PROCEDURE;
DECLARE I BYTE;

  RXTA$TRIES = RXTA$TRIES + 1;
  IF RXTA$TRIES > MAX$RXTA$TRIES THEN
    DO;
      TXTR = FALSE;
      RXTA = FALSE;
      CALL SRVC$TAB$H$SKP(2);
      RXTA$TRIES = 0;
    END;

  IF ((TX01NE - TX01NS) >= DATA$GRAM$SIZE) OR (TX01NS > TX01NE) THEN
    DO;
      I = 12;
      DO WHILE I <= 19;
        CALL HEX$ASC(TX01TB(TX01NS + I),(I-12)*2);
        I = I + 1;
      END;
      CALL SNDSEQ(.TP$55A, SIZE(TP$55A));
      CALL SNDSEQ(.CRLF, SIZE(CRLF));
      CALL SNDSEQ(.TEMP1, SIZE(TEMP1));
      CALL SNDSEQ(.TX01TB(TX01NS + 56), TCP$DATA$SIZE);
      IF (TRTA AND (NOT TXTR) AND (NOT RXTA) AND
          (NOT RXTR) AND (NOT TXTA)) THEN
        DO;
          TXTR = TRUE;
          CALL SNDSEQ(.TP$57, SIZE(TP$57));
          CALL SCOUT(TR);
        END;
      IF (NOT TRTA) THEN
        TRANS$1$RDY = TRUE;
    END;

END  READ$TXTAB;

/*****
* DATE:          25 SEP 84
* VERSION:       1.0
* NAME:          READ$RXTAB
* MODULE NUMBER: 4.0
* FUNCTION:      Reads the local receive tables
* INPUTS:        None
* OUTPUTS:       None
* GLOBAL VARIABLES USED:  RX01NE,RX01SZ,DATA$GRAM$SIZE,RX01TB
*                                RX01NS
* GLOBAL VARIABLES CHANGED:  None
*****/

```

SOURCE CODE FOR ISHOST.SRC, 15 AUG 85

```

* MODULES CALLED:      HEX$ASC, SNDSEQ, SRVC$TAB$H$SKP
* CALLING MODULES:      main
* AUTHOR:      Capt C.T. Childress
* HISTORY:      1.0 - Capt C. T. Childress
* *****/
READ$RXTAB:      PROCEDURE;
DECLARE I BYTE;

IF ((RX01NE - RX01NS) >= DATA$GRAM$SIZE) OR (RX01NS > RX01NE) THEN
DO:
  I = 12;
  DO WHILE I <= 19;
    CALL HEX$ASC(RX01TB(RX01NS + I).(I-12)*2);
    I = I + 1;
  END;
  CALL SNDSEQ(.TP$56A, SIZE(TP$56A));
  CALL SNDSEQ(.CRLF, SIZE(CRLF));
  CALL SNDSEQ(.TEMP1, SIZE(TEMP1));
  CALL SNDSEQ(.RX01TB(RX01NS + 56), TCP$DATA$SIZE);
  CALL SRVC$TAB$H$SKP(1);
END;

```

END READ\$RXTAB;

```

/*****
* DATE:      15 AUG 85
* VERSION:    1.1
* NAME:      LOAD
* MODULE NUMBER: 3.4
* FUNCTION:    This procedure places a message in the TX01TB
               table
* INPUTS:      TABLE$PTR, DEST$UNID, DEST$HOST
* OUTPUTS:      NONE
* GLOBAL VARIABLES USED:  MSGUM, THIS$COUNTRY$CODE, THIS$UNID$NBR
               CHAN$NUM, COUNTRY$CODE
* GLOBAL VARIABLES CHANGED:  MSGNUM
* MODULES CALLED:  SNDSEQ
* CALLING MODULES:  READ$LINE
* AUTHOR:      Capt Mark Weber
* HISTORY:      1.1 - Capt Mark Weber -15 AUG 85 -configured for ISIS
               1.0 - Capt C. T. Childress -25 SEP 85 configured for
               CP/M
* *****/
LOAD: PROCEDURE(TABLE$PTR, DEST$UNID, DEST$HOST);

DECLARE (INDEX, TABLE$PTR) ADDRESS;
DECLARE (DEST$UNID, DEST$HOST) BYTE;

```

SOURCE CODE FOR ISHOST.SRC, 15 AUG 85

DECLARE LCOXTB BASED TABLEPTR (1) BYTE;

```
DO INDEX = 0 TO (DATA$GRAM$SIZE - 1);
  LCOXTB(INDEX) = 0;
END;
```

/* VARIOUS CONSTANTS PER THE TCP/IP SPEC */

```
LCOXTB(0) = 4BH;
LCOXTB(3) = 80H;
LCOXTB(6) = 40H;
LCOXTB(8) = 3CH;
LCOXTB(9) = 6;
LCOXTB(20) = 82H;
LCOXTB(21) = 0BH;
```

/* IP SOURCE HEADER */

```
LCOXTB( 12) = THIS$COUNTRY$CODE AND OFH;
/* CT = 0, CC = THIS$COUNTRY$CODE */
LCOXTB( 13) = (ROL(THIS$UNID$NBR, 4) AND OFOH) OR
(ROR(((CHAN$NUM*56) AND OFOH), 4) AND OFH);
/* NC = THIS$UNID$NBR, HC(H) = CHAN$NUM * 56 */
LCOXTB( 14) = 7H OR ROL(((CHAN$NUM*56) AND OFH), 4) AND OFOH;
/* HC(L) = CHAN$NUM * 56, PC(2) = 7 */
LCOXTB( 15) = 77H; /* PC(1) = 7, PC(0) = 7 */
```

/* IP DESTINATION HEADER */

```
LCOXTB( 16) = THIS$COUNTRY$CODE AND OFH;
/* CT = 0, CC = THIS$COUNTRY$CODE */
LCOXTB( 17) = (ROL(DEST$UNID, 4) AND OFOH) OR
(ROR((DEST$HOST AND OFOH), 4) AND OFH);
/* NC = DEST$UNID, HC(H) = DEST$HOST */
LCOXTB( 18) = 7H OR ROL((DEST$HOST AND OFH), 4) AND OFOH;
/* HC(L) = DEST$HOST, PC(2) = 7 */
LCOXTB( 19) = 77H; /* PC(1) = 7, PC(0) = 7 */
```

```
CALL MOVE(TCP$DATA$SIZE, MESSAGE, LCOXTB( 56));
IF MSGNUM > 9 THEN
  MSGNUM = 0;
```

LCOXTB(86) = '0' + MSGNUM;

MSGNUM = MSGNUM + 1;

CALL SNDSEQ(.TP\$54, SIZE(TP\$54));

END LOAD;

```
/* *****
* DATE: 15 AUG 85
* VERSION: 1.1
* NAME: READ$LINE
* MODULE NUMBER: 3.0
* FUNCTION: Reads a line of data from the host console and
* *****
```

SOURCE CODE FOR ISHOST.SRC, 15 AUG 85

```

*
* then loads the specified number of datagrams into the
* transmit table according to the specified path
*
* INPUTS:
*   None
* OUTPUTS:
*   None
* GLOBAL VARIABLES USED:
*   TRTA, BUFFER, TX01TB, TX01NE, DEST$HOST$CODE
*   CHAN$NUM, FOREVER, DEST$NET$CODE
* GLOBAL VARIABLES CHANGED:
*   BUFFER, TRTA, DEST$NET$CODE, DEST$HOST$CODE
* MODULES CALLED:
*   SNDSEQ, READ$CON, ASC$HEX, HEX$ASC
* CALLING MODULES:
*   main
* AUTHOR: Capt Mark Weber
* HISTORY:
*   1.1 - Capt Mark Weber - adapted BUFFER to ISIS READ,
*         WRITE calls
*   1.0 - Capt C. T. Childress
* *****/
READ$LINE: PROCEDURE;
  DECLARE TABLE$PTR ADDRESS,
    (I, COUNT) BYTE;
  I = 0;
  CALL SNDSEQ(.MSG1, SIZE(MSG1));
  BUFFER(0) = 'N';
  CALL READ$CON;
  IF (BUFFER(0) = 'Y') OR (BUFFER(0) = 'y') THEN
    DO;
      CALL SNDSEQ(.MSG1A, SIZE(MSG1A));
      BUFFER(0) = 'N';
      CALL READ$CON;
      IF (BUFFER(0) = 'Y') OR (BUFFER(0) = 'y') THEN
        TRTA = TRUE;
      ELSE
        TRTA = FALSE;
      CALL SNDSEQ(.MSG3, SIZE(MSG3));
      CALL READ$CON;
      CHAN$NUM = ASC$HEX(BUFFER(0));
      IF (CHAN$NUM >= 1) AND (CHAN$NUM <= 4) THEN
        DO;
          TABLE$PTR = .TX01TB(TX01NE);
          CALL SNDSEQ(.MSG5, SIZE(MSG5));
          CALL READ$CON;
          DEST$NET$CODE = ASC$HEX(BUFFER(0));
          CALL SNDSEQ(.MSG6, SIZE(MSG6));
          CALL READ$CON;
          DEST$HOST$CODE = 0;
          COUNT = BUFFER(0);
          DO I = 0 TO 1;

```

SOURCE CODE FOR ISHOST.SRC, 15 AUG 85

```

DEST$HOST$CODE = ROL(DEST$HOST$CODE, 4);
DEST$HOST$CODE = DEST$HOST$CODE OR ASC$HEX(BUFFER(1));

END;

CALL SNDSEQ(.MSG4, SIZE(MSG4));
CALL READ$CON;
IF (BUFFER(0) >= '1') AND (BUFFER(0) <= '9') THEN
  DO I = 1 TO ASC$HEX(BUFFER(0));
    CALL LOAD(TABLE$PTR, DEST$NET$CODE, DEST$HOST$CODE);
    CALL LD$TAB$H$SKP(2);
    TABLE$PTR = .TX01TB(TX01NE);
  END;
END;

END;

CALL SNDSEQ(.TP$50, SIZE(TP$50));
CALL HEX$ASC(CHAN$NUM, 0);
CALL SNDSEQ(.TEMP1, 2);
CALL SNDSEQ(.TP$51A, SIZE(TP$51A));
CALL HEX$ASC(DEST$NET$CODE, 0);
CALL SNDSEQ(.TEMP1, 2);
CALL SNDSEQ(.TP$52A, SIZE(TP$52A));
CALL HEX$ASC(DEST$HOST$CODE, 0);
CALL SNDSEQ(.TEMP1, 2);

CALL SNDSEQ(.MSG2, SIZE(MSG2));
BUFFER(0) = 'N';
CALL READ$CON;

IF (BUFFER(0) = 'V') OR (BUFFER(0) = 'Y') THEN
  FOREVER = FALSE;
ELSE
  FOREVER = TRUE;
END;

END READ$LINE;

/*****
* DATE: 25 SEP 84
* VERSION: 1.0
* NAME: LOOP2
* MODULE NUMBER: 7.0
* FUNCTION: back to the network for validating iSBC 544 operation
* INPUTS: None
* OUTPUTS: None
* GLOBAL VARIABLES USED: None
* GLOBAL VARIABLES CHANGED: None
* MODULES CALLED: TRANS$1, RCV$1
* CALLING MODULES: main
* AUTHOR: Capt C.T. Childress
*****/

```


SOURCE CODE FOR ISHOST.SRC, 15 AUG 85

```

* HISTORY:
*          1.0 - Capt C. T. Childress
*****/

LOOP2: PROCEDURE;
    CALL TRANS$1;
    CALL RCV$1;
END LOOP2;

/***** THIS IS THE MAIN BODY OF THE PROGRAM *****/
/* *****/
/*****/

BEGIN:
    CALL INIT;
    CALL SNDSEQ(.STARTUP$HDR, SIZE(STARTUP$HDR));
    CALL SNDSEQ(.INIT$MSG, SIZE(INIT$MSG));
    CALL SNDSEQ(.CR,LF), 2);

    FOREVER = TRUE;
    CALL READ$LINE;
    CALL SCLRCM;
    DO WHILE FOREVER;
        CALL READ$XTAB;
        IF TRTA THEN
            CALL CHK$RXTA;
            IF TRANS$1$RDY THEN
                CALL LOOP2;
            IF TRTA THEN
                CALL CHK$RXTA;
            CALL READ$RXTAB;
            IF SCMCHK <> 0 THEN
                CALL CHK$RXTA;
            CALL READ$LINE;
        END;
        CALL SNDSEQ(.MSG7, SIZE(MSG7));
        CALL EXIT;
    END MAIN;

/***** THE END *****/

```

APPENDIX K

HOST I/O MODULES

1. Intel 210 CP/M I/O Module (assembly) K-2
2. Intel 230 ISIS I/O Module (assembly) K-11
3. Intel 230 ISIS I/O Module (PL/M) K-15

1. Intel 210 CP/M O/P Module (assembly)

APPENDIX K

HOST I/O MODULES

1. Intel 210 CP/M I/O Module (assembly). K-2
2. Intel 230 ISIS I/O Module (assembly). K-11
3. Intel 230 ISIS I/O Module (PL/M) K-16

ISIS ASSEMBLY SOURCE CODE FOR HOST1.ASM, 20 JUL 85

```

*****
LINK80 SUBROUTINES
*****

; In directory as TSBS.ASM, SBS.ASM, or SBS.MOD
; Set up for California Computer System 2810 CPU
; (at 4 MHz) which uses the WD or NS 8250 UART
; This software module was developed by Capt C. T. (Tom)
; Childress on his own time and with his own resources
; for use with his own CP/M system communications
; program. It is a proven module of software. This
; software and his system are used in his thesis as a
; software validation tool for the SBC 544 board.
; Any other use without prior written permission from
; Capt Childress is strictly prohibited.

;
; Vers 1.0, 10 May 82
; Vers 1.1, 26 Sep 82
; Vers 1.2, 19 Oct 82
; Vers 1.3, 11 Feb 83, New video board driver, 19.2 & 38.4 kbps
; Vers 1.4, 24 Mar 83, Changed XON/XOFF toggle for Terminal transmit file
; Vers 1.5, 3 Jul 83, Added Echo to remote for terminal mode
; Vers 1.6, 30 Jul 84, Deleted the XON/OFF and Echo for use with higher
; speeds in the terminal mode file transfer mode
; Vers 1.7, 17 Aug 84, Modified to work with the UNID II 15BC 544 to
; support the Internet Protocol (IP) datagram exchange
; running on the 15BC 544.
; Vers 1.8, 20 Jul 85, Modified to work with the Intel system II/210
; operation under CPM and with 8251 USART

; EQUATES
;
; SDATA EQU 0F4H ;Serial data port
; SCONT EQU SDATA+1 ;Serial command port
; SSTAT EQU SDATA+1 ;Serial status port
;
; 8253 Counter ports and control mode
; COUNTER EQU 0F3H ;Counter mode port
; CNTBAUD EQU 0F0H ;Counter 0 baud port
; CNTMODE EQU 00110110B ;Counter mode3, counter 0 command
;
; RXRDY EQU 00000010B ;Rx data available
; TXMTY EQU 0000001B ;Tx buffer empty
; ERROR EQU 00110000B ;Error bit mask
; DSRMSK EQU 10000000B ;DSR bit mask
; SYNDET EQU 01000000B ;Sync detect mask
; RESET EQU 01000000B ;Software reset
;
; BRFO EQU 00000000B ;Bit Rate Factor x 0 = synchronous

```

ISIS ASSEMBLY SOURCE CODE FOR HOST1.ASM, 20 JUL 85

```

BRF1 EQU 0000001B ;Bit Rate Factor x 1
BRF16 EQU 0000010B ;Bit Rate Factor x 16
BRF64 EQU 0000011B ;Bit Rate Factor x 64
;
; comm EQU 0011011B ;tx, tx enabled
;
; CR EQU 00H
; LF EQU 0AH
; XON EQU 11H
; XOF EQU 13H
; bdos0 equ 5
; wvec equ 0
;
; cseg ;Relative assembly for ASM80
;
; Start of jump table is public
;
; public sinit, sclrcm, scmchk, scmin, scmout
;
; Console, BDOS, and warm boot are public
;
; public const, conin, conout, bdos, exit
;
; Jump table
;
; START:
; sinit: JMP INIT ;Initialization
; sclrcm: JMP CLRCM ;Clear comm channel
; scmchk: JMP CMCHK ;Check channel
; scmin: JMP CMIN ;Input byte
; scmout: JMP CMOUT ;Output byte
;
; CP/M BDOS and BIOS call vectored jumps
;
; bdos: jmp bdos0 ;Bdos call
; exit: jmp wvec ;warm boot
;
; const: lhd const ;Bios console status
; pchl
; conin: lhd conin0 ;Bios console input
; pchl
; conout: lhd conout0 ;Bios console output
; pchl
; const0: dw 0 ;Bios console status vector
; conin0: dw 0 ;Bios console input vector
; conout0: dw 0 ;Bios console output vector
;
;
; Menu initialization routines
;

```

```

INIT:
    lhd    wvec + 1 ;Calculate the bios console
    lxi d,3 ; status, input, output vectors
    dad d
    shld   const0
    dad d
    shld   conin0
    dad d
    shld   cnout0

;
    LXI D,MSG5 ;CCS signon
    MVI C,9
    CALL 5
    MVI C,1 ;Get keystroke
    CALL 5
    INITS: LXI D,CLEAR ;Clear screen
    MVI C,9
    CALL 5
    LXI D,MENU
    MVI C,9 ;Display menu selections
    CALL 5
    LXI D,INBUF ;Get selection
    MVI C,10
    CALL 5
    LXI H,INBUF+1 ;Check for null select
    MOV A,M
    ORA A
    JZ INIT1 ;No select, go initialize
    INX H ;Get feature
    MOV A,M
    CALL ULCAS ;Make upper case
    CPI 'A' ;Data rate?
    LXI D,RATE ;Get save addr
    JNZ INIT2 ;No
    JMP SAVIT ;Save it
    INIT2: CPI 'B' ;Data bits?
    LXI D,BITS1 ;Get save addr
    JNZ INIT3 ;No
    JMP SAVIT ;Save it
    INIT3: CPI 'C' ;Stop bits?
    LXI D,BITS2 ;Get save addr
    JNZ INIT4 ;No
    JMP SAVIT ;Save it
    INIT4: CPI 'D' ;Parity
    LXI D,PAR ;Get save addr
    JNZ INITS ;No
    SAVIT: INX H ;Go past delimiter
    INX H ;Get selection
    MOV A,M
    CALL VALNUM ;Is it a valid number?

```

```

JC INITS ;Repeat if not
STAX D ;Save it
JMP INITS ;Another selection?

;
INIT1: LDA RATE ;Get data rate
SUI '0' ;Make binary
CPI 10 ;Check for more than allowed
JP INITS ;Repeat if so
RLC ;Double for two bytes/entry
LXI H,BTBL ;Get data rate table addr
MVI D,0 ;Init D
MOV E,A ;Put offset in E
DAD D ;Add to table addr
; (hl) now has the pointer to the data rate divisor
;
LDA BITS1 ;Get data bits
CPI '1' ;Check for 7 bit
JZ BITS7 ;It's 7 bit
CPI '2' ;Check for 8 bit
JNZ INITS ;Error, do again
LDA BIT8 ;Get 8 bit mask
JMP BITS4 ;Continue
BITS7: LDA BIT7 ;Get 7 bit mask
BITS4: MOV B,A ;Save it
;
LDA BITS2 ;Get stop bits
CPI '1' ;Check for 1 bit
JZ STPA ;It's 1 bit
CPI '2' ;Check for 2 bits
JNZ INITS ;Error, start over
LDA STP2 ;Get 2 bit mask
JMP STPB ;Continue
STPA: LDA STP1 ;Get 1 bit mask
STPB: ORA B ;Add 7/8 bit mask
MOV B,A ;Resave it
;
LDA PAR ;Get parity
CPI '1' ;Check for none
JZ PARN ;It's no parity
CPI '2' ;Check for even
JZ PARE ;It's even parity
CPI '3' ;Check for odd
JNZ INITS ;Error, do again
LDA ODPAR ;Get odd par mask
JMP PARX ;Continue
PARN: LDA NOPAR ;Get no par mask
JMP PARX ;Continue
PARE: LDA EVPAR ;Get even par mask
PARX: ORA B ;Add in 7/8 and 1/2 bits
MOV B,A ;Resave

```



```

;
; MVI A,BRF16 ;Add in the bit rate factor
; ORA B ;to the rest of the mode byte
; MOV B,A ;Resave the mode byte
;
; User provided initialization routine (Baud rate, etc.)
; ****UART INITIALIZATION ROUTINE****
; For WD or NS 8250 UART: VR 1.7
; For 8251/A UART: VR 1.8
;
; RATE CODE MULTIPLIER
; 110 2BAH OCEH ;reset the 8251 USART
; 300 040H OCFH ;load the 8253 counter for mode 3
; 1200 010H OCFH ;and lsb followed by msb
; 2400 020H OCEH ;baud rate factor for 9600
; 4800 010H OCEH ;lsb
; 9600 008H OCEH ;msb
; 19200 004H OCEH
;
; MVI A,RESET ;reset the 8251 USART
; OUT SCNT
; MVI A,CNTMODE ;load the 8253 counter for mode 3
; OUT COUNTER ;and lsb followed by msb
;
; LXI H,008H ;baud rate factor for 9600
; MOV A,L ;lsb
; OUT CNTBAUD ;
; MOV A,H ;msb
; OUT CNTBAUD ;
;
; MOV A,B ;LOAD THE MULTIPLIER TO THE USART!
; OUT SCNT ;x 16 multiplier ( OCEH DEFAULT )
; MVI A,commd ;enable tx, rx, and set rts, dtr
; OUT SCNT
; RET
;
; Clear communications channel
; Do not output anything to the remote!!!!
;
; CLRCM: IN SDATA
; RET
;
; Check communications channel for anything
; Return with accumulator=0 if nothing ready,
; non-zero if a byte is waiting
;
; CMCHK:
; IN SSTAT
; ANI RXRDY
; RET

```

```

; Input byte from remote terminal
;
CMIN:
    IN  SSTAT ;Added for speed
    ANI RXRDV ;added for speed
    JZ  CMIN  ;added for speed
    IN  SDATA ;Get good data
    RET

; Output byte to remote computer
;
CMOUT:
OTWT:
    IN  SSTAT ;Get the transmit status
    ANI TXMTV
    JZ  OTWT  ;Not ready yet
    MOV A,C ;It is now, send the byte
    OUT SDATA
    RETURN: RET ;Default return

; ULCAS: CPI 'a' ;Less than lower case a?
    RM ;Yes
    CPI 'z'+1 ;> lower case z?
    RP ;Yes
    SUI ; ;No, clear lower case bit
    RET

; VALNUM: CPI '0' ;Less than 0?
    STC
    RM ;Carry set; not valid number
    CPI ; ;More than 9?
    STC
    RP ;Carry set; not valid number
    CMC
    RET ;Carry cleared; valid number

; NOTE: 1ah is the clear screen code for the Soroc 120
;        1bh,1h for the H-19
;
MSG5: DB 1BH,62H,'Configured for the CCS 2810 CPU (at 4 MHz), 20 Jul 85',CR,LF
DB 'Working with the UNID II 1SBC 544 Internet Protocol (IP)',CR,LF
DB 'to simulate an actual host system.',CR,LF
DB 'Hit key to continue $',CR,LF

; MENU:
DB 'Feature Name Choice Option',CR,LF,LF
DB 'a Data Rate'
RATE: DB '7'
DB '0- 110, 1- 150, 2- 300',CR,LF

```

```

ISIS ASSEMBLY SOURCE CODE FOR HOST1.ASM, 20 JUL 85

DB ,
DB ,
DB ,
DB ,

;
DB , b Data Bits
BITS1: DB '2'
DB , 1-7 bit, 2-8 bit',CR,LF,LF
;
DB , c Stop Bits
BITS2: DB '1'
DB , 1-1 bit, 2-2 bits',CR,LF,LF
;
DB , d Parity
PAR: DB '1'
DB , 1-None, 2-Even, 3-Odd',CR,LF,LF
;
CHOIC: DB 'Show choice by letter, space or comma, number, and CR',CR,LF
DB 'Hit CR when menu selections complete: $'
;
;Rate table for 2.45 MHz xtal
;
BTBL: DB 1396 MOD 256 ;110 bps
DB 1396/256
DB 1024 MOD 256 ;150
DB 1024/256
DB 512 MOD 256 ;300
DB 512/256
DB 256 MOD 256 ;600
DB 256/256
DB 128 MOD 256 ;1200
DB 128/256
DB 64 MOD 256 ;2400
DB 64/256
DB 32 MOD 256 ;4800
DB 32/256
DB 16 MOD 256 ;9600
DB 16/256
DB 8 MOD 256 ;19200
DB 8/256
DB 4 MOD 256 ;38400
DB 4/256
;
BIT7: DB 00001000B ;7 bit mask
BIT8: DB 00001100B ;8 bit mask
STP1: DB 01000000B ;1 stop bit
STP2: DB 11000000B ;2 stop bits
NOPAR: DB 0 ;No parity
EVPAR: DB 00110000B ;Even parity
ODPAR: DB 00010000B ;Odd parity
;

```

ISIS ASSEMBLY SOURCE CODE FOR HOST1.ASM, 20 JUL 85

CLEAR: DB 1BH,62H,'\$' ;1AH: String to clear screen (Soroc IQ 120)
; ;1BH,2H

INBUF: DB 5 ;Input buffer, max # chars

DB 0 ;# chars in buff

DS 5 ;Storage for chars

; DB 'end S&S'

; END

;

2. Intel 230 ISIS I/O Module (assembly)

```

ISIS ASSEMBLY SOURCE CODE FOR HOST2.ASM, 21 JUL 85

$TITLE('HOST2.ASM: I/O MODULE FOR INTEL 230 UNDER ISIS')
;*****
; LINK80 SUBROUTINES
;*****
;
; FILE NAME: HOST2.ASM
; This software module was developed by Capt C. T. (Tom)
; Childress and modified by Capt Mark Weber for operation
; on the ISIS SERIES II/III systems. The program
; initializes 8251/8251A USARTS to 9600 baud, parity
; disabled, 8 bit ASCII, Full Duplex. This was used as a
; software validation tool for the SBC 544 board.
;
; Vers 1.0, 10 May 82
; Vers 1.1, 26 Sep 82
; Vers 1.2, 19 Oct 82
; Vers 1.3, 11 Feb 83, New video board driver, 19.2 & 38.4 kbps
; Vers 1.4, 24 Mar 83, Changed to 4MHz, ABCD drives
; Vers 1.5, 3 Jul 83, Added XON/XOFF toggle for Terminal transmit file
; mode, Added Echo to remote for terminal mode
; Vers 1.6, 30 Jul 84, Deleted the XON/OFF and Echo for use with higher
; speeds in the terminal mode file transfer mode
; Vers 1.7, 17 Aug 84, Modified to work with the UNID II ISBC 544 to
; support the Internet Protocol (IP) datagram exchange
; running on the ISBC 544.
; Vers 1.8, 20 Jul 85, Modified to support ISIS system, Filed under
; HOST1.ASM.
; Vers 1.9, 21 Jul 85, Version 1.8 with CPM menu removed, Filed under
; HOST2.ASM
;
; EQUATES
;
; SDATA EQU 0F4H ;Serial data port
; SCONT EQU SDATA+1 ;Serial command port
; SSTAT EQU SDATA+1 ;Serial status port
;
; 8253 Counter ports and control mode
; COUNTER EQU 0F3H ;Counter mode port
; CNTBAUD EQU 0F0H ;Counter 0 baud port
; CNTMODE EQU 0110110B ;Counter mode3, counter 0 command
;
; RXRDY EQU 00000010B ;Rx data available
; TXMTY EQU 0000001B ;Tx buffer empty
; ERROR EQU 0111000B ;Error bit mask
; DSRMSK EQU 1000000B ;DSR bit mask
; SYSNDET EQU 0100000B ;Sync detect mask
; RESET EQU 0100000B ;Software reset
;
; BRFO EQU 00000000B ;Bit Rate Factor x 0 = synchronous
; BRF1 EQU 00000001B ;Bit Rate Factor x 1

```

ISIS ASSEMBLY SOURCE CODE FOR HOST2.ASM, 21 JUL 85

```

BRF16 EQU 00000010B ;Bit Rate Factor x 16
BRF63 EQU 00000011B ;Bit Rate Factor x 64
;
; commd EQU 00110111B ;tx, tr enabled
;
; CR EQU 0DH
; LF EQU 0AH
; XON EQU 11H
; XOF EQU 13H
;
; cseg ;Relative assembly for ASM80
;
; Start of jump table is public
;
; public sinit, sclrcm, scmchk, scmin, scmout
;
; Jump table
;
; START:
; sinit: JMP INIT ;Initialization
; sclrcm: JMP CLRCM ;Clear comm channel
; scmchk: JMP CMCHK ;Check channel
; scmin: JMP CMIN ;Input byte
; scmout: JMP CMOUT ;Output byte
;
;
; INIT:
;
;
; User provided initialization routine (Baud rate, etc.)
; ***UART INITIALIZATION ROUTINE***
; For WD or NS 8250 UART; VR 1.7
; FOR 8251/A USART VR 1.8
;
; RATE CODE MULTIPLIER
; 110 2BAH OCEH
; 300 040H OCFH
; 1200 010H OCFH
; 2400 020H OCEH
; 4800 010H OCEH
; 9600 008H OCEH
; 19200 004H OCEH
;
; MVI A,RESET ;reset the 8251 USART
; OUT SCNT
; MVI A,CNTMODE ;load the 8253 counter for mode 3
; OUT COUNTER ;and lsb followed by msb
;
; LXI H,008H ;LOAD THE BAUD RATE INTO THE COUNTER!
; MOV A,L ;baud rate factor for 9600
; ;lsb

```

```

OUT CNTBAUD
MOV A,H
OUT CNTBAUD
LOAD THE MULTIPLIER TO THE USART!
MVI A,0CEH
OUT SCNT
MVI A,commd
OUT SCNT
RET

; Clear communications channel
; Do not output anything to the remote!!!!
;
CLRCM: IN SDATA
RET

; Check communications channel for anything
; Return with accumulator=0 if nothing ready,
; non-zero if a byte is waiting
;
CMCHK:
IN SSTAT
ANI RXRDY
RET

; Input byte from remote terminal
;
CMIN:
IN SSTAT
ANI RXRDY
JZ CMIN
IN SDATA
RET

; Output byte to remote computer
;
CMOUT:
OTWT:
IN SSTAT
ANI TXMTY
JZ OTWT
MOV A,C:It is now, send the byte
OUT SDATA
RETURN: RET

;
;
; END
;

```


3. Intel 230 ISIS I/O Module (PL/M)

```

$TITLE('PLM80 DRIVERS FOR HOST SYSTEM 220/230 UNDER ISIS, 18 SEP 85')
* DATE: 18 SEP 85
* VERSION: 1.0
* TITLE: PLM80 I/O driver software
* FILENAME: HOST3.SRC
* COORDINATOR: Capt Mark W. Weber
* PROJECT: UNIT II
* OPERATING SYSTEM: INTEL SYSTEM III/230 UNDER ISIS
* LANGUAGE: PL/M 80
* USE: This file must be linked with ISHOST to operate the channel 1
      (TTY) input/output port on the Intel SYSTEM 220/230. This module
      replaces the assembly module used with ISHOST (either HOST1.ASM,
      or HOST2.ASM)
* CONTENTS:
*   SINIT - initializes the serial port
*   SCLRCM - clears the serial port
*   SCMCHK - checks the status on the serial port
*   SCMIN - input routine for the serial port
*   SCMOUT - output routine for the serial port
*   FUNCTION: This module operates the Intel SYSTEM 220/230 micro computer
      channel 1 (TTY) port. The port uses an 8251A USART. Changing
      the port addresses on this module may allow other systems
      to use these procedures if they also use an 8251/8251A USART.
* HISTORY: 1.0 Mark Weber - implemented X.25 datalink procedures
*****
HOST3: DO:
  DECLARE SDATA LITERALLY '0F4H';
  DECLARE SCNT LITERALLY '0F5H';
  DECLARE SSTAT LITERALLY '0F5H';
  DECLARE COUNTER LITERALLY '0F3H';
  DECLARE CNTBAUD LITERALLY '0F0H';
  DECLARE CNTMODE LITERALLY '036H';
  DECLARE COMMAND LITERALLY '037H';
  DECLARE RXRDY LITERALLY '02H';
  DECLARE TXMTY LITERALLY '01H';
  ERROR LITERALLY '038H';
  DSRMAK LITERALLY '080H';
  SYSNDET LITERALLY '040H';
  RESET LITERALLY '040H';

  DECLARE BRFO LITERALLY '00H';
  BRF1 LITERALLY '01H';
  BRF16 LITERALLY '02H';
  BRF64 LITERALLY '03H';

  DECLARE FALSE LITERALLY '0H';
  DECLARE TRUE LITERALLY '0FFH';
  /*
  /* SERIAL PORT DATA ADDRESS
  /* SERIAL PORT CONTROL ADDRESS
  /* SERIAL PORT STATUS ADDRESS
  /* 8255 COUNTER MODE ADDRESS
  /* 8255 COUNTER BAUD RATE ADDRESS
  /* 8255 MODE COMMAND
  /* 8251A MODE COMMAND
  /* RX DATA READY
  /* TX BUFFER EMPTY
  /* ERROR BIT MASK
  /* DSR BIT MASK
  /* SYNC DETECT BIT MASK
  /* SOFTWARE RESET
  /* BIT RATE FACTOR X 0 OR SYNC
  /* BIT RATE FACTOR X 1
  /* BIT RATE FACTOR X 16
  /* BIT RATE FACTOR X 64
  /*
  BAUD RATE TABLE
  /*

```

SOURCE CODE FOR HOST3.SRC DRIVERS FOR HOST SYSTEM 220/230 UNDER ISIS, 18 SEP 85

```

/*****
*
*   RATE          CODE          MULTIBLIER
*   110           2BAH          OCEH
*   300           040H          OCFH
*   1200          010H          OCFH
*   2400          020H          OCEH
*   4800          010H          OCEH
*   9600          008H          OCEH
*   19200         004H          OCEH
*
*   DATE:         18 SEP 85
*   VERSION:      1.2
*   NAME:         SINIT
*
*   MODULE NUMBER: This procedure initializes the communications
*   DESCRIPTION:   channel for full duplex operation.
*
*   PASSED VARIABLES:
*   RETURNS:      None
*   GLOBAL VARIABLES USED:
*   GLOBAL VARIABLES CHANGED:
*   MODULES CALLED:
*   CALLING MODULES:
*   AUTHOR:       Mark Weber
*   HISTORY:      1.2 Capt Mark Weber - original PL/M 80 version
*                1.1 Capt Mark Weber - system 230/220 version
*                1.0 Capt C.T. Chidress - original assembly version
*
*   SINIT: PROCEDURE PUBLIC;
*
*   OUTPUT(SCONT) = RESET;
*   OUTPUT(COUNTER) = CNTMODE;
*   OUTPUT(CNTBAUD) = 08H;
*   OUTPUT(CNTBAUD) = 00H;
*   OUTPUT(SCONT) = OCEH;
*   OUTPUT(SCONT) = COMMAND;
*
*   END SINIT;
*
*   DATE:         18 SEP 85
*   VERSION:      1.2
*   NAME:         SCLRM
*
*   MODULE NUMBER: This procedure clears the communications channel.
*   DESCRIPTION:
*   PASSED VARIABLES:
*   RETURNS:      None
*   GLOBAL VARIABLES USED:
*   GLOBAL VARIABLES CHANGED:
*   MODULES CALLED:
*
*   INPUT$BYTE
*   None
*   None
*   None
*   None
*
*   LOW BITS */
*   HIGH BITS */
*   USART MULTIPLIER */
*   USART COMMAND BYTE */
*****/

```

SOURCE CODE FOR HOST3.SRC DRIVERS FOR HOST SYSTEM 220/230 UNDER ISIS, 18 SEP 85

```

* CALLING MODULES:
* AUTHOR: Mark Weber
* HISTORY: 1.2 Capt Mark Weber - original PL/M 80 version
*          1.1 Capt Mark Weber - system 230/220 version
*          1.0 Capt C.T. Childress - original assembly version
*          *****/
SCLRCM: PROCEDURE BYTE PUBLIC;

```

```

    DECLARE INPUT$BYTE BYTE;

```

```

    INPUT$BYTE = INPUT(SDATA);
    RETURN INPUT$BYTE;

```

```

END SCLRCM;

```

```

/*****
* DATE: 18 SEP 85
* VERSION: 1.2
* NAME: SCMCHK
* MODULE NUMBER: This procedure checks the status of the USART.
* PASSED VARIABLES: None
* RETURNS: PORT$STATUS
* GLOBAL VARIABLES USED: None
* GLOBAL VARIABLES CHANGED: None
* MODULES CALLED: None
* CALLING MODULES:
* AUTHOR: Mark Weber
* HISTORY: 1.2 Capt Mark Weber - original PL/M 80 version
*          1.1 Capt Mark Weber - system 230/220 version
*          1.0 Capt C.T. Childress - original assembly version
*          *****/
SCMCHK: PROCEDURE BYTE PUBLIC;

```

```

    DECLARE PORT$STATUS BYTE;

```

```

    PORT$STATUS = SHR((INPUT(SSSTAT) AND RXRDY), 1);
    RETURN PORT$STATUS;

```

```

END SCMCHK;

```

```

/*****
* DATE: 18 SEP 85
* VERSION: 1.2
* NAME: SCMIN
* MODULE NUMBER:
* DESCRIPTION: This procedure is the input routine for the channel.
* PASSED VARIABLES: None
* RETURNS: DATA$INPUT
*

```

SOURCE CODE FOR HOST3.SRC DRIVERS FOR HOST SYSTEM 230/230 UNDER ISIS, 18 SEP 85

```

* GLOBAL VARIABLES USED:      None
* GLOBAL VARIABLES CHANGED:   None
* MODULES CALLED:             None
* CALLING MODULES:
* AUTHOR:      Mark Weber
* HISTORY:     1.2 Capt Mark Weber - original PL/M 80 version
*             1.1 Capt Mark Weber - system 230/220 version
*             1.0 Capt C.T. Childress - original assembly version
*****
SCMIN: PROCEDURE BYTE PUBLIC;

```

```

    DECLARE (DATA$INPUT, READY) BYTE;

```

```

    READY = FALSE;
    DO WHILE NOT READY;
        READY = SHR((INPUT(SSTAT) AND RXRDV), 1);
    END;
    DATA$INPUT = INPUT(SDATA);
    RETURN DATA$INPUT;

```

```

END SCMIN;

```

```

/*****
* DATE:      18 SEP 85
* VERSION:   1.2
* NAME:      SCMOUT
*
* MODULE NUMBER: This procedure clears the communications channel.
*
* PASSED VARIABLES:
* RETURNS:      None
*
* GLOBAL VARIABLES USED:
* GLOBAL VARIABLES CHANGED:
* MODULES CALLED:
* CALLING MODULES:
* AUTHOR:      Mark Weber
* HISTORY:     1.2 Capt Mark Weber - original PL/M 80 version
*             1.1 Capt Mark Weber - system 230/220 version
*             1.0 Capt C.T. Childress - original assembly version
*****
SCMOUT: PROCEDURE (CHAR$OUT) PUBLIC;

```

```

    DECLARE (CHAR$OUT, READY) BYTE;

```

```

    READY = FALSE;
    DO WHILE NOT READY;
        READY = INPUT(SSTAT) AND TXMTY;
    END;
    OUTPUT(SDATA) = CHAR$OUT;

```

SOURCE CODE FOR HOST3.SRC DRIVERS FOR HOST SYSTEM 220/230 UNDER ISIS, 18 SEP 85 PAGE 6

END SCOUT;

END HOST3;

/*-----END MODULE HOST3.SRC-----*/

APPENDIX L

SBC 544 Simulation Software

SIM544.SRC contains the format modifications to the SBC 544 simulation software used to validate the frame and packet header formats made on the embedded SBC 544 software.

SOURCE CODE FOR SIM544.SRC. 30 SEP 85
 \$TITLE('SBC 544 LOCAL NETWORK TEST SIMULATION, 30 SEP 1985')

```

/*****
* DATE: 20 SEP 85
* VERSION: 1.3
* TITLE: ISO layer 3B SBC 544 SIMULATION WITH THE X.25 PACKET FORMAT
* FILENAME: SIM544.SRC
* COORDINATOR: Capt Mark W. Weber
* PROJECT: UNID II
* OPERATING SYSTEM: 8080/8085 CPU (embedded software)
* LANGUAGE: PL/M 80
* USE: This file requires no includes and is used to validate the
*       the upper network level functions performed by the SBC 544 via
*       simulation. This file has the X.25 packet format implemented.
*       No other changes have been made between this file and the file
*       NEWLOD.LOC. This file must be linked with PLM80.LIB and SYSTEM.LIB.
*       The file may be linked using LNK544(SIM544, PLM80.LIB, SYSTEM.LIB).
*
* CONTENTS: MAIN - main processing routine
*            DELAY - causes a selectable delay
*            HEX$ASC - converts HEX to ASCII
*            ASC$HEX - converts ASCII to HEX
*            VALID$HEX - validates HEX numbers
*            IN$INT - initializes the necessary hardware
*            INIT$TAB - initializes the necessary tables and pointers
*            TRNMIT$PKT - initializes a data port for packet transmission
*            ERR$CHK - processes system errors
*            SNDSEQ - writes messages system console
*            LD$TAB$H$SKP - updates the transmit tables
*            SRVC$TAB$H$SKP - updates the receive tables
*            RXIN$1 - USART 0 (channel 1) monitor input routine
*            TXOUT$1 - USART 0 (channel 1) monitor output routine
*            BUILD$I$PACKET - formats a message into a data packet
*            DET$ADDR - procedure determines the destination of local data
*            DET$ADDR$NL - determines the destination of data from the
*                          datalink layer
*            MOVE$LL - moves a packet between local host channels
*            MOVE$NL - moves a packet between the local and network channels
*            SERVICE$LOOP - software loop to return packets to receive
*                          channel
*            ROUTE$IN - processes incoming packets from the transport layer
*            LOAD - places a packet in transmit table LC0xTB
*            READ$XTAB - reads the transmit tables for data to send
*            READ$SLCTAB - reads the local table for message
*            UPDATE$LCPTR - updates the channel pointer within a table
*            READ$SLINE - reads user responses
*            main - calls procedures to simulate operational SBC 544 in UNID
*            Implements the simulation of the SBC 544 in the UNID II using
*            the TCP/IP protocol as defined by Phister. This SBC handles
*            the functions of ISO layer 3B which processes the IP protocol
*
* FUNCTION:
*
*****/

```


SOURCE CODE FOR SIMS44.SRC. 30 SEP 85

```

* and handles a multi-host requirements.
* HISTORY: 1.2 Mark Weber - implemented X.25 packet format
*          1.1 C.T. Childress - 2 OCT 84 - original implementation
*          1.0 P. Philster - original
*          *****
/*****
MAIN: DO;
READ:  PROCEDURE (AFTN, BUFFER, COUNT, ACTUAL, STATUS) EXTERNAL;
END READ;
WRITE: PROCEDURE (AFTN, BUFFER, COUNT, STATUS) EXTERNAL;
END WRITE;
EXIT:  PROCEDURE EXTERNAL;
END EXIT;
ERROR: PROCEDURE (ERRNUM) EXTERNAL;
END ERROR;
DECLARE ACTUAL ADDRESS;
DECLARE STATUS ADDRESS;
DECLARE BUFFER (128) BYTE;
DECLARE R$CONN LITERALLY '1';
DECLARE W$CONN LITERALLY '0';
DECLARE ERNUM ADDRESS;
DECLARE CRLF(*) BYTE DATA(ODH, OAH);
DECLARE MESSAGE(*) BYTE DATA(ODH, OAH,
'THIS IS THE TEST MESSAGE
ASCII(*) BYTE DATA('0123456789ABCDEF');
DECLARE CHAN$NUM ADDRESS;
DECLARE
L$R1$DEST$ERR LITERALLY '000H'; /* LOCAL ROUTE$IN ERROR */
L$R0$DEST$ERR LITERALLY '01H'; /* LOCAL ROUTE$OUT ERROR */
/* THE FOLLOWING ARE UNID DEFINED VARIABLES */
/* NOTE: THESE VARIABLES MAY CHANGE DEPENDING ON THE
SOFTWARE CONFIGURATION USED WITHIN THE DELNET */
DECLARE
DATA$GRAM$SIZE LITERALLY '128'; /* NUMBER OF BYTES FROM HOST */
PACKET$SIZE LITERALLY '138'; /* DATA PACKET + HEADER */
PACKET$IN$TABLE LITERALLY '10';
STAT$NBR LITERALLY '20'; /* STATUS ENTRIES IN STATB */
DATA$TABLE$SIZE LITERALLY '1280'; /* DATAGRAM * NBR OF DATAGRAMS */

```

SOURCE CODE FOR SIM544.SRC. 30 SEP 85

```

PACKET$TABLE$SIZE LITERALLY '1380', /* PACKET * NBR OF PACKETS */
TCP$DATA$SIZE LITERALLY '72', /* TCP DATA SIZE */

/* FOLLOWING ARE NETWORK DEFINED VARIABLES */
/* NOTES: 1. THIS$UNID$NBR MUST REFLECT WHICH UNID THIS IS.
2. THIS$COUNTRY$CODE MUST REFLECT THE AREA TO WHICH
THIS UNID IS LOCATED.
3. MAX$COUNTRY$CODE WILL INDICATE WHICH COUNTRY CODES
ARE CURRENTLY OPERATIONAL. CC=0000 IS RESERVED FOR
THE DELNET MONITOR.
4. MAX$NETWORK$CODE WILL INDICATE HOW MANY UNIDS ARE
CURRENTLY OPERATIONAL WITHIN A PARTICULAR COUNTRY.
5. FOR DETAILED INFORMATION ON THE ABOVE REFER TO
PHISTER'S THESIS, APPENDIX D. */

THIS$UNID$NBR LITERALLY '02', /* UNIQUE ADDRESS FOR THIS UNID */
THIS$COUNTRY$CODE LITERALLY '01', /* CC WHERE THIS UNID RESIDES */
MAX$COUNTRY$CODE LITERALLY '01', /* INDICATES COUNTRY$CODES IN USE */
MAX$NETWORK$CODE LITERALLY '03'; /* INDICATES UNIDS OPERATIONAL IN NET */

/******
/* DEFINITIONS FOR THE LOCAL SERIAL INPUT/OUTPUT CARD AND 86/12 PPI
/******
DECLARE
  CNTRL$B255 LITERALLY '0CEH', /* 8255 CONTROL PORT ON 86/12 */
  ASIN LITERALLY '0CBH', /* I/O INPUT PORT ADDRESS */
  BSOUT LITERALLY '0CAH', /* I/O OUTPUT PORT ADDRESS */
  C$CNTRL LITERALLY '0CCH', /* I/O CONTROL PORT ADDRESS */
  ICW1$OR$OCW2 LITERALLY '0COH', /* PORT ADDRESS FOR PIC */
  ICWS LITERALLY '0C2H', /* PORT ADDRESS FOR PIC MODE */
  RCV$STATE LITERALLY '00010110B', /* MODE INSTRUCTION FOR */
  TRANS$STATE LITERALLY '00110111B', /* 8251 USART RCV INT */
  ROTATES$PRIORITY$SET LITERALLY '10100000B'; /* 8251 RCV & XMIT INT */
  ROTATES$PRIORITY$SET LITERALLY '10100000B'; /* ROTATES PRIORITY */
  ROTATES$PRIORITY$SET LITERALLY '10100000B'; /* FOR EQUAL PRIORITY TO */
  ROTATES$PRIORITY$SET LITERALLY '10100000B'; /* ALL I/O PORTS */

```

```

/******
/* ADDITIONAL GENERAL DECLARES NEEDED FOR THIS PROGRAM
/******

```

```

DECLARE
  BAUD$LSB BYTE,
  BAUD$MSB BYTE,
  NUM$BYTES$SENT ADDRESS,
  TRANS$1$RDY BYTE,
  TRANS$2$RDY BYTE,
  TRANS$3$RDY BYTE,
  TRANS$4$RDY BYTE,
  MSGNUM BYTE;

```

```

SOURCE CODE FOR SIM544.SRC.          30 SEP 85

DECLARE      TEMP1(*) BYTE INITIAL('000000000000000000'),
             TEMP2(*) BYTE INITIAL('000000000000000000');

DECLARE      BYTESSENT$1 BYTE,
             BYTESSENT$2 BYTE,
             BYTESSENT$3 BYTE,
             BYTESSENT$4 BYTE;

DECLARE      J          BYTE;

DECLARE      HSKP$ERR ADDRESS;

/*****
/* DATA TABLES USED IN THIS PROGRAM
*****/

DECLARE      LC01TB (DATA$TABLE$SIZE) BYTE,
             LC01NS ADDRESS,
             LC01NE ADDRESS,
             LC01SZ ADDRESS,

             LC02TB (DATA$TABLE$SIZE) BYTE,
             LC02NS ADDRESS,
             LC02NE ADDRESS,
             LC02SZ ADDRESS,

             LC03TB (DATA$TABLE$SIZE) BYTE,
             LC03NS ADDRESS,
             LC03NE ADDRESS,
             LC03SZ ADDRESS,

             LC04TB (DATA$TABLE$SIZE) BYTE,
             LC04NS ADDRESS,
             LC04NE ADDRESS,
             LC04SZ ADDRESS,

             TX01TB(DATA$TABLE$SIZE) BYTE,
             TX01NS ADDRESS,
             TX01NE ADDRESS,
             TX01SZ ADDRESS,

             TX02TB(DATA$TABLE$SIZE) BYTE,
             TX02NS ADDRESS,
             TX02NE ADDRESS,
             TX02SZ ADDRESS,

             TX03TB(DATA$TABLE$SIZE) BYTE,
             TX03NS ADDRESS,
             TX03NE ADDRESS,

```

TX03SZ ADDRESS,

TX04TB(DATA\$TABLE\$SIZE) BYTE,
TX04NS ADDRESS,
TX04NE ADDRESS,
TX04SZ ADDRESS,

LCNTNS ADDRESS,
LCNTNE ADDRESS,
LCNTSZ ADDRESS,

NLCNS ADDRESS,
NLCNE ADDRESS,
NLCSZ ADDRESS,

STATTB (STAT\$NBR) BYTE;

/* DECLARATIONS FOR FLAGS AND TABLES IN SYSTEM MEMORY */

DECLARE

INTERUPT BYTE, /* 88/45 INTERRUPT BYTE */
(SPARE1, SPARE2, SPARE3) BYTE, /* SPARE LOCATIONS */
(TO\$HOST, TO\$HOST\$BUSY) BYTE, /* BOOLEAN FLAGS */
(TO\$HOSTNS, TO\$HOSTNE) ADDRESS, /* TABLE ADDRESS */
NTLCNB (PACKET\$TABLE\$SIZE) BYTE, /* TABLE OF 10 PACKETS */
(TO\$NET, TO\$NET\$BUSY) BYTE, /* BOOLEAN FLAGS */
(TONETNS, TONETNE) ADDRESS, /* TABLE ADDRESS */
LCNTTB (PACKET\$TABLE\$SIZE) BYTE; /* TABLE OF 10 PACKETS */

/* MISCELLANEOUS DECLARATIONS */

DECLARE FOREVER BYTE;
DECLARE BUSY LITERALLY 'OFFH';
TRUE LITERALLY 'OFFH';
FALSE LITERALLY '00H';
NMBR\$MSK LITERALLY '07H';
CR LITERALLY '00H';
LF LITERALLY '0AH';
SOURCE LITERALLY '12';
DESTIN LITERALLY '16';
ESC LITERALLY '1BH';

DECLARE

DESTINATION ADDRESS, /* DESTINATION OF THE PACKET */
DESTINATION\$ADDRESS BYTE, /* DESTIN ADDR OF DATAGRAM */
SOURCE\$ADDRESS BYTE; /* SOURCE ADDR OF DATAGRAM */

/* INTERNAL VARIABLES USED IN THIS MODULE */

DECLARE

STARTUP\$HDR(*) BYTE DATA(CR, LF,

SOURCE CODE FOR SIM544.SRC. 30 SEP 85

UNID II #2 LOCAL OS',CR,LF,
 VERS 1.2, 30 SEP 85',CR,LF,
 EXECUTING ',CR,LF);

/* THE FOLLOWING TEST POINTS ARE USED TO FOLLOW THE DATA WITHIN THE UNID */

```

DECLARE
  'TP$1(*) BYTE DATA(CR,LF,
    ENTERING INIT$STAB PROCEDURE');
  'TP$2(*) BYTE DATA(CR,LF,
    ENTERING INIT$SHTAB PROCEDURE');
  'TP$3(*) BYTE DATA(CR,LF,
    ENTERING INVT PROCEDURE');
  'TP$4(*) BYTE DATA(CR,LF,
    STARTING ROUTE$IN-ROUTE$OUT LOOP');
  'TP$5(*) BYTE DATA(CR,LF,
    ENTERING ROUTE$IN PROCEDURE');
  'TP$6(*) BYTE DATA(CR,LF,
    DATA LOCATED IN LOCAL CHANNEL-1');
  'TP$7(*) BYTE DATA(CR,LF,
    DATA IS LC01TB TO TX0XTB TRANSFER');
  'TP$8(*) BYTE DATA(CR,LF,
    DATA IS LC01TB TO LCNTTB TRANSFER');
  'TP$9(*) BYTE DATA(CR,LF,
    ERROR OCCURED IN LOCAL CHANNEL-1 IN-PROCESSING');
  'TP$10(*) BYTE DATA(CR,LF,
    DATA LOCATED IN LOCAL CHANNEL-2');
  'TP$11(*) BYTE DATA(CR,LF,
    DATA IS LC02TB TO TX0XTB TRANSFER');
  'TP$12(*) BYTE DATA(CR,LF,
    DATA IS LC02TB TO LCNTTB TRANSFER');
  'TP$13(*) BYTE DATA(CR,LF,
    ERROR OCCURED IN LOCAL CHANNEL-2 IN PROCESSING');
  'TP$14(*) BYTE DATA(CR,LF,
    DATA LOCATED IN LOCAL CHANNEL-3');
  'TP$15(*) BYTE DATA(CR,LF,
    DATA IS LC03TB TO TX0XTB TRANSFER');
  'TP$16(*) BYTE DATA(CR,LF,
    DATA IS LC03TB TO LCNTTB TRANSFER');
  'TP$17(*) BYTE DATA(CR,LF,
    ERROR OCCURED IN LOCAL CHANNEL-3 IN PROCESSING');
  'TP$18(*) BYTE DATA(CR,LF,
    DATA LOCATED IN LOCAL CHANNEL-4');
  'TP$19(*) BYTE DATA(CR,LF,
    DATA IS LC04TB TO TX0XTB TRANSFER');
  'TP$20(*) BYTE DATA(CR,LF,
    DATA IS LC04TB TO LCNTTB TRANSFER');
  'TP$21(*) BYTE DATA(CR,LF,
    ERROR OCCURED IN LOCAL CHANNEL-4 IN PROCESSING');
  'TP$22(*) BYTE DATA(CR,LF,
    ENTERING ROUTE$OUT PROCEDURE');
  'TP$22

```

SOURCE CODE FOR SIM544.SRC. 30 SEP 85

```

DECLARE
  TP$23(*) BYTE DATA(CR,LF,
    OUTGOING DATA IS IS, LCLCTB');
  TP$24(*) BYTE DATA(CR,LF,
    DATA IN LCOXTB DESTINED FOR LOCAL CHANNEL-1');
  TP$25(*) BYTE DATA(CR,LF,
    DATA IN LCOXTB DESTINED FOR LOCAL CHANNEL-2');
  TP$26(*) BYTE DATA(CR,LF,
    DATA IN LCOXTB DESTINED FOR LOCAL CHANNEL-3');
  TP$27(*) BYTE DATA(CR,LF,
    DATA IN LCOXTB DESTINED FOR LOCAL CHANNEL-4');
  TP$28(*) BYTE DATA(CR,LF,
    ERROR OCCURED IN LCOXTB OUT$PROCESSING');
  TP$29(*) BYTE DATA(CR,LF,
    OUTGOING DATA IS IN, NTLCTB');
  TP$30(*) BYTE DATA(CR,LF,
    DATA IN NTLCTB DESTINED FOR LOCAL CHANNEL-1');
  TP$31(*) BYTE DATA(CR,LF,
    DATA IN NTLCTB DESTINED FOR LOCAL CHANNEL-2');
  TP$32(*) BYTE DATA(CR,LF,
    DATA IN NTLCTB DESTINED FOR LOCAL CHANNEL-3');
  TP$33(*) BYTE DATA(CR,LF,
    DATA IN NTLCTB DESTINED FOR LOCAL CHANNEL-4');
  TP$34(*) BYTE DATA(CR,LF,
    ERROR OCCURED IN NTLCTB TO TXOXTB OUT-PROCESSING');
  TP$35(*) BYTE DATA(CR,LF,
    END OF ROUTE$IN-ROUTE$OUT LOOP');
  TP$36(*) BYTE DATA(CR,LF,
    HAVE EXITED ROUTE$IN-ROUTE$OUT LOOP and returned to ISIS',CR,LF);

DECLARE
  TP$40(*) BYTE DATA(CR,LF,
    Packet moved from LCNTTB to NTLCTB');
  TP$50(*) BYTE DATA(CR,LF,
    Channel Number = ');
  TP$51(*) BYTE DATA(CR,LF,
    Destination = ');
  TP$51A(*) BYTE DATA(CR,LF,
    Destination Network = ');
  TP$51B(*) BYTE DATA(CR,LF,
    Destination Control Code = ');
  TP$51C(*) BYTE DATA(CR,LF,
    Destination Country Code = ');
  TP$51D(*) BYTE DATA(CR,LF,
    Lobits = ');
  TP$51E(*) BYTE DATA(CR,LF,
    Hibits = ');
  TP$52(*) BYTE DATA(CR,LF,
    Destination Address = ');
  TP$52B(*) BYTE DATA(CR,LF,
    Source Address = ');
  TP$52A(*) BYTE DATA(CR,LF,

```

SOURCE CODE FOR SIM544.SRC. 30 SEP 85

```

'TP$52A      Destination Host = ' );
DECLARE TP$53(*) BYTE DATA(CR,LF,
'TP$53
DECLARE TP$53A(*) BYTE DATA(CR,LF,
'TP$53A      Destination Network Code > Max Network');
DECLARE TP$53AA(*) BYTE DATA(CR,LF,
'TP$53AA     Source Network Code > Max Network');
DECLARE TP$53B(*) BYTE DATA(CR,LF,
'TP$53B      Country Code <> This Country - Sent to UNID 0');
DECLARE TP$53C(*) BYTE DATA(CR,LF,
'TP$53C      Control Code <= 0');
DECLARE TP$54(*) BYTE DATA(CR,LF,
'TP$54      Loaded test datagram in LCOXTB');
DECLARE TP$55A(*) BYTE DATA(CR,LF,
'TP$55A      Reading test datagram from TX01TB');
DECLARE TP$55B(*) BYTE DATA(CR,LF,
'TP$55B      Reading test datagram from TX02TB');
DECLARE TP$55C(*) BYTE DATA(CR,LF,
'TP$55C      Reading test datagram from TX03TB');
DECLARE TP$55D(*) BYTE DATA(CR,LF,
'TP$55D      Reading test datagram from TX04TB');
DECLARE TP$55E(*) BYTE DATA(CR,LF,
'TP$55E      Reading test datagram from LC01TB');
DECLARE TP$55F(*) BYTE DATA(CR,LF,
'TP$55F      Reading test datagram from LC02TB');
DECLARE TP$56A(*) BYTE DATA(CR,LF,
'TP$56A      Reading test datagram from LC03TB');
DECLARE TP$56B(*) BYTE DATA(CR,LF,
'TP$56B      Reading test datagram from LC04TB');
DECLARE TP$56C(*) BYTE DATA(CR,LF,
'TP$56C      Reading test datagram from LC05TB');
DECLARE TP$56D(*) BYTE DATA(CR,LF,
'TP$56D      Reading test datagram from LC06TB');

DECLARE MSG1(*) BYTE DATA(CR,LF,
'Do you want to load the test message? ');
DECLARE MSG2(*) BYTE DATA(CR,LF,
'Do you want to stop the test? ');
DECLARE MSG3(*) BYTE DATA(CR,LF,
'Load into which host channel (1,2,3,4)? ');
DECLARE MSG4(*) BYTE DATA(CR,LF,
'How many datagrams (0 - 9)? ');
DECLARE MSG5(*) BYTE DATA(CR,LF,
'Destination network code (1,2,3) = ');
DECLARE MSG6(*) BYTE DATA(CR,LF,
'Destination host code (0 - FFH) = ');

```

```

/*****
* DATE: 25 SEP 84
* VERSION: 1.0
* NAME: DELAY
* MODULE NUMBER: Not yet implemented
* FUNCTION: The purpose of this procedure is to add a
*****/

```

SOURCE CODE FOR SIM544.SRC, 30 SEP 85

```

*      delay to parts of the initialization procedure that
*      are time dependent. Delay is approximately in
*      milliseconds.
*      INPUTS:
*      OUTPUTS:      None
*      GLOBAL VARIABLES USED:      None
*      GLOBAL VARIABLES CHANGED:      None
*      MODULES CALLED:      TIME (PLM80.LIB)
*      CALLING MODULES:      None yet
*      AUTHOR:      Capt C. T. Childress
*      HISTORY:
*      1.0 - Capt C. T. Childress -25 SEP 85 configured for
*      CP/M
*****/
DELAY:PROCEDURE PUBLIC;
  DECLARE I BYTE;
  DO I=1 TO 40;
    CALL TIME(250); /* TIME IS A BUILT IN FUNCTION OF PLM86 WHICH CAUSES */
                  /* A DELAY BASED ON THE NUMBER IN PARENS */
  END;
END DELAY;

```

```

/*****
*      DATE:      25 SEP 84
*      VERSION:      1.0
*      NAME:      HEX$ASC
*      MODULE NUMBER:      4.6
*      FUNCTION:      Converts a HEX number into its ascii characters
*                      for printing to console
*      INPUTS:      VAL - a hex number
*      OUTPUTS:      HEX$ASC
*      GLOBAL VARIABLES USED:      ASCII - array of legal hex characters
*                      I - position in string
*      GLOBAL VARIABLES CHANGED:      TEMP1 - ASCII output array
*      MODULES CALLED:      None
*      CALLING MODULES:      SNDSEQ, READ$TX$TAB
*      AUTHOR:      Capt C.T. Childress
*      HISTORY:
*      1.0 - Capt C. T. Childress -25 SEP 84
*****/
HEX$ASC: PROCEDURE(VAL, I);
  DECLARE (VAL, I) BYTE;

```

```

  TEMP1(I) = ASCII(SHR(VAL,4) AND 0FH);
  TEMP1(I+1) = ASCII(VAL AND 0FH);

```

```

END HEX$ASC;

```

```

/*****

```


SOURCE CODE FOR SIM544.SRC. 30 SEP 85

```

* DATE: 25 SEP 84
* VERSION: 1.0
* NAME: ASC$HEX
* MODULE NUMBER: 4.2
* FUNCTION: Converts ASCII characters to their HEX equivalent
: valient number.
* INPUTS: C - the character
* OUTPUTS: ASC$HEX
* GLOBAL VARIABLES USED: None
* GLOBAL VARIABLES CHANGED: None
* MODULES CALLED: None
* CALLING MODULES: READLINE
* AUTHOR: Capt C.T. Childress
* HISTORY: 1.0 - Capt C. T. Childress -25 SEP 84
*****/
ASC$HEX: PROCEDURE(C) BYTE;
DECLARE C BYTE;

```

```

IF (C >= '0' AND C <= '9') THEN
  RETURN (C-30H);
ELSE
  IF (C >= 'A' AND C <= 'F') THEN
    RETURN (C-37H);

```

RETURN C;

END ASC\$HEX;

```

/*****
* DATE: 25 SEP 84
* VERSION: 1.0
* NAME: VALID$HEX
* MODULE NUMBER: Not yet used
* FUNCTION: Validates a char to be changed to hex
* INPUTS: H - char
* OUTPUTS: VALIDHEX - boolean
* GLOBAL VARIABLES USED: ASCII - array of valid ascii char
* GLOBAL VARIABLES CHANGED: None
* MODULES CALLED: None
* CALLING MODULES: None yet
* AUTHOR: Capt C.T. Childress
* HISTORY: 1.0 - Capt C. T. Childress -25 SEP 84
*****/
VALID$HEX: PROCEDURE(H) BYTE;
DECLARE (H,I) BYTE;

```

```

DO I = 0 TO LAST(ASCII);
  IF H = ASCII(I) THEN

```

```

END;
RETURN FALSE;

```

```

END VALID$HEX;

```

```

/*****
* DATE: 24 SEP 84
* VERSION: 1.0
* NAME: INVINT
* MODULE NUMBER: 1.0
* FUNCTION: Initializes the software for handling all global
* variables, tables, pointers and the communications
* channel.
* INPUTS: None
* OUTPUTS: None
* GLOBAL VARIABLES USED: BYTES$SENT$1, BYTES$RECV$1,
* TRANS1$RDY, TRANS2$RDY, TRANS3$RDY, TRANS4$RDY
* same as above
* MODULES CALLED: None
* CALLING MODULES: main
* AUTHOR: Capt C.T. Childress
* HISTORY:
* 1.0 - Capt C. T. Childress -25 SEP 84
*****/
INVINT: PROCEDURE;

```

```

/* INITIALIZE 8255 PROGRAMMABLE PERIPHERAL INTERFACE (PPI) */
/* SET BAUD RATES FOR LOCAL CHANNEL USARTS */
/* SET PIT #1, REGISTER #1 FOR USART #2 BAUD RATE */
/* SET PIT #1, REGISTER #2 FOR USART #3 BAUD RATE */
/* SET PIT #2, REGISTER #0 FOR USART #4 BAUD RATE */
/* INITIATE RESET ON ALL USARTS */
/* INITIALIZE THE FOUR USARTS TO: 2 STOP BITS, NO PARITY,
/* 8 BIT CHARACTERS, AND 16X BAUD RATE */
/* INITIALIZE THE 8259 PROGRAMMABLE INTERRUPT CONTROLLER */

```

```

MSGNUM = 0;
BYTES$SENT$1 = 0;
BYTES$SENT$2 = 0;
BYTES$SENT$3 = 0;
BYTES$SENT$4 = 0;
TRANS1$RDY = TRUE;
TRANS2$RDY = TRUE;
TRANS3$RDY = TRUE;
TRANS4$RDY = TRUE;

```

```

END INVINT;

```

30 SEP 85

SOURCE CODE FOR SIM544.SRC.

```

/*****
*   DATE:          2 OCT 84
*   VERSION:       1.1
*   NAME:          INIT$TAB
*   MODULE NUMBER: 3.0
*   DESCRIPTION:   Initializes the pointers and tables used with
*                 the SBC 544 software.
*   PASSED VARIABLES:  None
*   RETURNS:        None
*   GLOBAL VARIABLES USED:
*
*       LC01NS, LC02NS, LC03NS, LC04NS, LC01NE,
*       LC02NE, LC03NE, LC04NE, LC01SZN, LC02SZ, LC03ST,
*       LC04SZ, LC01NS, LC02NS, LC03NS, LC04NS, TX01NE,
*       TX02SZ, TX01NS, TX02NS, TX03NS, TX04NS, TX01NE,
*       TX02NE, TX03NE, TX04NE, TX01SZN, TX02SZ, TX03ST,
*       TX04SZ, STATTB, STAT$NBR
*
*   GLOBAL VARIABLES CHANGED: All the above
*   MODULES CALLED:          None
*   CALLING MODULES:        main
*   AUTHOR:                  Capt C.T. Childress
*
*   HISTORY:
*       1.1 Capt C.T. Childress - original 544 version
*
*****
INIT$TAB: PROCEDURE;
DECLARE IX ADDRESS;

LCNTNS = 0;
LCNTNE = 0;
LCNTSZ = PACKET$TABLE$SIZE;

NTLCNS = 0;
NTLCNE = 0;
NTLCNZ = PACKET$TABLE$SIZE;

LC01NS = 0;
LC01NE = 0;
LC01SZ = DATA$TABLE$SIZE;

LC02NS = 0;
LC02NE = 0;
LC02SZ = DATA$TABLE$SIZE;

LC03NS = 0;
LC03NE = 0;
LC03SZ = DATA$TABLE$SIZE;

LC04NS = 0;
LC04NE = 0;
LC04SZ = DATA$TABLE$SIZE;

```

30 SEP 85

SOURCE CODE FOR SIM544.SRC,

```
TX01NS = 0;
TX01NE = 0;
TX01SZ = DATA$TABLE$SIZE;
```

```
TX02NS = 0;
TX02NE = 0;
TX02SZ = DATA$TABLE$SIZE;
```

```
TX03NS = 0;
TX03NE = 0;
TX03SZ = DATA$TABLE$SIZE;
```

```
TX04NS = 0;
TX04NE = 0;
TX04SZ = DATA$TABLE$SIZE;
```

```
IX = 0;
DO WHILE IX < STAT$NBR;
  STAT$B(IX) = 0;
  IX = IX + 1;
END;
```

END INIT\$TAB;

```
/* ***** 2 OCT 84 ***** */
* DATE: 1.1
* VERSION: 1.1
* NAME: TRNMIT$PKT
* MODULE NUMBER: 6.3.2
* DESCRIPTION: This enables data port for packet transmission.
* Presently the procedure serves as a null procedure for
* simulation purposes.
* PASSED VARIABLES: TABLE
* RETURNS: None
* GLOBAL VARIABLES USED: TRANS$STATE
* GLOBAL VARIABLES CHANGED: None
* MODULES CALLED: None
* CALLING MODULES: MOVE$LL
* AUTHOR: Capt C.T. Childress
* HISTORY:
* 1.1 Capt C.T. Childress - original 544 version
/*
TRNMIT$PKT: PROCEDURE(PORT$NUM);
  DECLARE PORT$NUM BYTE;
  DISABLE;
  DO CASE PORT$NUM;
```

SOURCE CODE FOR SIMS44.SRC. 30 SEP 85

```

;
OUTPUT(PORT$1) = TRANS$STATE;
OUTPUT(PORT$2) = TRANS$STATE;
OUTPUT(PORT$3) = TRANS$STATE;
OUTPUT(PORT$4) = TRANS$STATE;

```

```

END;
ENABLE;

```

```

END TRNMIT$PKT;
*/

```

```

/*****
* DATE: 15 AUG 85
* VERSION: 1.1
* NAME: ERR$CHK
* MODULE NUMBER: 4.3
* FUNCTION: Checks the status of ISIS call procedures and
             for an orderly exit incase of error
* INPUTS: None
* OUTPUTS: None
* GLOBAL VARIABLES USED: STATUS
* GLOBAL VARIABLES CHANGED: None
* MODULES CALLED: ERROR, EXIT
* CALLING MODULES: SNDSEQ, READ$CON
* AUTHOR: Capt Mark Weber
* HISTORY:
           1.1 - Capt Mark Weber -15 AUG 85 changed from CPM exit to
           ISIS exit procedures
           1.0 - Capt C. T. Childress 25 SEP 84
*****/
ERR$CHK: PROCEDURE;
IF STATUS <> 0 THEN
DO;
CALL ERROR(STATUS);
CALL EXIT;
END;

```

```

END ERR$CHK;

```

```

/*****
* DATE: 15 AUG 85
* VERSION: 1.1
* NAME: SNDSEQ
* MODULE NUMBER: 2.0
* FUNCTION: Sends a message to the system console
* INPUTS: MSG - the message address
*****/

```

SOURCE CODE FOR SIM544.SRC, 30 SEP 85

```

*      TOTAL - the count of the char in the buffer
*      None (Text to console output)
*      STATUS
* GLOBAL VARIABLES USED:
* GLOBAL VARIABLES CHANGED:
* MODULES CALLED:
* CALLING MODULES:
*   main, READ$LINE, LOAD, READ$RX$TAB,
*   READ$TX$TAB, RCV$1
* AUTHOR:  Capt Mark Weber
* HISTORY:
* HISTORY:
*   1.1 - Capt Mark Weber -15 AUG 85 -configured for ISIS
*   1.0 - Capt C. T. Childress -25 SEP 84 configured for
*         CP/M
* *****/
SNDSEQ: PROCEDURE( MSG, TOTAL);
DECLARE MSG ADDRESS;
DECLARE TOTAL ADDRESS;

CALL WRITE( W$CONN, MSG, TOTAL, .STATUS);
CALL ERR$CHK;

```

END SNDSEQ;

```

/*****
* DATE: 25 SEP 84
* VERSION: 1.0
* NAME: LD$TAB$H$SKP
* MODULE NUMBER: 6.5
* FUNCTION: This procedure housekeeps a specified buffer
*           after loading the user data from the host. The pro-
*           cedure determines which table to process, and advances
*           the NEX$EMPTY$BYTE address by one PACKET$SIZE, and if
*           necessary wraps the data around to the beginning of
*           the queue.
* INPUTS: TABLE - address of the table to process
* OUTPUTS: None
* GLOBAL VARIABLES USED:
*   LCOINE (2, 3, 4), LCNTNE, LCNTSZ,
*   LCOISZ (2, 3, 4), NTLCNE, NTLCSZ, HSKP$ERR
*   TXOINE (2, 3, 4), TXOISZ (2, 3, 4)
* GLOBAL VARIABLES CHANGED:
*   LCOINE (2, 3, 4), TXOINE (2, 3, 4),
*   HSKP$ERR, NTLCNE, LCNTNE
* MODULES CALLED:
*   NONE
* CALLING MODULES:
*   READ$LINE, READ$TXTB, SERVICES$LOOP,
*   ROUTE$IN
* AUTHOR:  Capt C. T. Childress -25 SEP 84
* HISTORY:
*   1.0 - Capt C. T. Childress -25 SEP 85 configured for
*         CP/M
* *****/

```

```
LD$TAB$H$SKP: PROCEDURE(TABLE);
DECLARE TABLE ADDRESS;
```

```
IF (TABLE >= 1 AND TABLE <= 11) THEN /* 6 FOR REAL, 11 FOR SIM */
DO CASE TABLE;
```

```
    ; /* CASE ZERO IS NULL */
```

```
DO; /* START CASE ONE */
    LC01NE = LC01NE + DATA$GRAM$SIZE; /* ADVANCE NEXT$EMPTY ADDRESS */
    IF LC01NE >= LC01SZ THEN
        LC01NE = 0;
END;
```

```
DO; /* START CASE TWO */
    LC02NE = LC02NE + DATA$GRAM$SIZE; /* ADVANCE NEXT$EMPTY ADDRESS */
    IF LC02NE >= LC02SZ THEN
        LC02NE = 0;
END;
```

```
DO;
    LC03NE = LC03NE + DATA$GRAM$SIZE;
    IF LC03NE >= LC03SZ THEN
        LC03NE = 0;
END;
```

```
DO;
    LC04NE = LC04NE + DATA$GRAM$SIZE;
    IF LC04NE >= LC04SZ THEN
        LC04NE = 0;
END;
```

```
    ; /* START OF CASE FIVE */

DO; /* START OF CASE SIX */
    LCNTNE = LCNTNE + PACKET$SIZE;
    IF LCNTNE >= LCNTSZ THEN
        LCNTNE = 0;
END;
```

```
DO; /* CASE SEVEN IS NULL */
    NTLCNE = NTLCNE + PACKET$SIZE;
    IF NTLCNE >= NTLCSZ THEN
        NTLCNE = 0;
END;
```

```
DO;
    TX01NE = TX01NE + DATA$GRAM$SIZE;
    IF TX01NE >= TX01SZ THEN
        TX01NE = 0;
```

SOURCE CODE FOR SIM544.SRC. 30 SEP 85

```

END;

DO;
  TX02NE = TX02NE + DATA$GRAM$SIZE;
  IF TX02NE >= TX02SZ THEN
    TX02NE = 0;
END;

DO;
  TX03NE = TX03NE + DATA$GRAM$SIZE;
  IF TX03NE >= TX03SZ THEN
    TX03NE = 0;
END;

DO;
  TX04NE = TX04NE + DATA$GRAM$SIZE;
  IF TX04NE >= TX04SZ THEN
    TX04NE = 0;
END;

END;

ELSE HSKP$ERR = HSKP$ERR + 1;

END LD$TAB$HSKP;

/*****
* DATE: 25 SEP 84
* VERSION: 1.0
* NAME: SRCV$TAB$HSKP
* MODULE NUMBER: 6.8
* FUNCTION: This procedure housekeeps a specified table
  after servicing removing a packet. The procedure
  determines which table to process, advances the next
  byte to service pointer, and performs any necessary
  wrap around requirements
* INPUTS: TAB - pointer to table to process
* OUTPUTS: None
* GLOBAL VARIABLES USED: LCOINS (2, 3, 4), LCNTNS, NTLCONS,
  TX0INS (2, 3, 4), HSKP$ERR, NTLCTB
* GLOBAL VARIABLES CHANGED: all above
* MODULES CALLED: None
* CALLING MODULES: READ$RX$TAB, READLINE, SERVICE$LOOP,
  ROUTES$IN
* AUTHOR: Capt C.T. Childress
* HISTORY: 1.0 - Capt C. T. Childress -25 SEP 85 configured for
  CP/M
*****/

```


SOURCE CODE FOR SIM544.SRC.

30 SEP 85

PAGE 19

SRVC\$TAB\$H\$K\$P: PROCEDURE(TAB) ;
DECLARE TAB ADDRESS;

IF (TAB >= 1 AND TAB <= 11) THEN /* 7 FOR REAL, 11 FOR SIM */
DO CASE TAB;

 ; /* CASE ZERO IS NULL */

DO;
 /* CASE ONE STARTS HERE */
 LC01NS = LC01NS + DATA\$GRAM\$SIZE;
 IF LC01NS >= LC01SZ THEN
 LC01NS = 0;
END;

DO;
 /* CASE TWO STARTS HERE */
 LC02NS = LC02NS + DATA\$GRAM\$SIZE;
 IF LC02NS >= LC02SZ THEN
 LC02NS = 0;
END;

DO;
 LC03NS = LC03NS + DATA\$GRAM\$SIZE;
 IF LC03NS >= LC03SZ THEN
 LC03NS = 0;
END;

DO;
 /* CASE FOUR STARTS HERE */
 LC04NS = LC04NS + DATA\$GRAM\$SIZE; /* ADVANCE NEXT SERVICE ADDRESS */
 IF LC04NS >= LC04SZ THEN
 LC04NS = 0;
END;

 ; /* CASE FIVE STARTS HERE */

DO;
 /* CASE SIX IS A NULL STATEMENT */
 LCNTNS = LCNTNS + PACKET\$SIZE;
 IF LCNTNS >= LCNTSZ THEN
 LCNTNS = 0;
END;

DO;
 /* CASE SEVEN STARTS HERE */
 NTLCNS = NTLCNS + PACKET\$SIZE; /* ADVANCE NEXT SERVICE ADDRESS */
 IF NTLCNS >= NTLCSZ THEN
 NTLCNS = 0;
END;

/* CASES 8, 9, 10, 11 FOR TX01(2,3,4) ARE NOT REQUIRED AS */
/* THE TX0XNS POINTERS ARE INCREMENTED AND CHECKED IN THE */
/* TRANSMIT DATAGRAM ROUTINES */

SOURCE CODE FOR SIM544.SRC. 30 SEP 85

```

DO;
  TX01NS = TX01NS + DATA$GRAM$SIZE;
  IF TX01NS >= TX01SZ THEN
    TX01NS = 0;
  END;

DO;
  TX02NS = TX02NS + DATA$GRAM$SIZE;
  IF TX02NS >= TX02SZ THEN
    TX02NS = 0;
  END;

DO;
  TX03NS = TX03NS + DATA$GRAM$SIZE;
  IF TX03NS >= TX03SZ THEN
    TX03NS = 0;
  END;

DO;
  TX04NS = TX04NS + DATA$GRAM$SIZE;
  IF TX04NS >= TX04SZ THEN
    TX04NS = 0;
  END;
END;

ELSE HSKP$ERR = HSKP$ERR + 1;
END SRV$TAB$HSKP;

```

```

/*****
* DATE: 20 SEP 85
* VERSION: 1.2
* NAME: BUILDS$PACKET
* MODULE NUMBER: 6.4
* DESCRIPTION: This procedure receives an address that
* indicates the local input buffer where the incoming
* host data is located, then formats a packet for
* transmission to the datalink layer. SIM544 uses
* Philster's protocol and only formats the first two of
* five bytes of the packe. Future implementations will
* require a change to the packet format.
* PASSED VARIABLES: TABLE$PRT, PORT
* RETURNS: None
* GLOBAL VARIABLES USED: LCNTTB, LCNTNE
* GLOBAL VARIABLES CHANGED: All the above
* MODULES CALLED: None
* CALLING MODULES: READ$LINE
* AUTHOR: Capt M. Weber
*
*****/

```

SOURCE CODE FOR SIM544.SRC, 30 SEP 85

```

* HISTORY: 1.2 Mark Weber - X.25 packter format modification
*          1.1 C. T. Childress - original 544 version
*****
BUILD$I$PACKET: PROCEDURE(TABLE$PTR) ;
    DECLARE TABLE$PTR ADDRESS,
        LCOXTB BASED TABLE$PTR(1) BYTE;

    LCNTTB(LCNTNE + 0) = 010H; /* GFI/LGCN */
    LCNTTB(LCNTNE + 1) = (SOURCE$ADDRESS AND OFOH) OR
        (SHL(DESTINATION$ADDRESS, 4) AND OFH); /* SRC, DEST FIELD LENGTH */
    LCNTTB(LCNTNE + 2) = 0; /* SEQUENCE NUMBER */
    LCNTTB(LCNTNE + 3) = 044H; /* SRC, DEST FIELD LENGTH */
    LCNTTB(LCNTNE + 4) = SOURCE$ADDRESS; /* SRC, DEST FIELD LENGTH */
    LCNTTB(LCNTNE + 5) = DESTINATION$ADDRESS; /* SRC, DEST FIELD LENGTH */
    LCNTTB(LCNTNE + 6) = 0; /* SRC, DEST FIELD LENGTH */
    LCNTTB(LCNTNE + 7) = 010H; /* SRC, DEST FIELD LENGTH */
    LCNTTB(LCNTNE + 8) = 0; /* SRC, DEST FIELD LENGTH */
    LCNTTB(LCNTNE + 9) = 0; /* SRC, DEST FIELD LENGTH */

```

```

CALL MOVE(DATA$GRAM$SIZE, TABLE$PTR, LCNTTB(LCNTNE+10));

```

```

CALL LD$TAB$H$K$P(6);

```

```

END BUILD$I$PACKET;

```

```

/*****
* DATE: 2 OCT 84
* VERSION: 1.1
* NAME: DET$ADDR
* MODULE NUMBER: 6.2
* DESCRIPTION: This module determines the destination of data from the local host. A pointer is passed to the routine which gives the location of the packet to be evaluated. The routine processes the control byte, the COUNTRY$CODE, and NETWORK$CODE of the packet.
*              An example of a destination address is 21
*              for UNID = 2, Channel = 1.
* PASSED VARIABLES: TABLE$PTR
* RETURNS: None
* GLOBAL VARIABLES USED: STATTB DESTINATION, DESTINATION$ADDRESS,
* GLOBAL VARIABLES CHANGED: All the ones above
* MODULES CALLED: None
* CALLING MODULES: ROUTE$IN
* AUTHOR: C.T. Childress
*****/

```

```

SOURCE CODE FOR SIMS44.SRC,          30 SEP 85

* HISTORY:  1.1 C.T. Childress - 30 SEP 84 - original SBC 544
*
* ***** implementation *****
DET$ADDR: PROCEDURE(TABLE$PTR);
DEclare LOBITS BYTE,
        HIBITS BYTE,
        CONTROL$CODE BYTE,
        COUNTRY$CODE BYTE,
        NETWORK$CODE BYTE,
        HOST$CODE BYTE,
        SRC$HOST$CODE BYTE,
        SRC$NET$CODE BYTE,
        SRC$CONTRY$CODE BYTE,
        SRC$CONTRY$CODE BYTE,
        TABLE$PTR ADDRESS,
        LCOXTB BASED TABLE$PTR(1) BYTE;

LOBITS = 0;
HIBITS = 0;
CONTROL$CODE = 0;
COUNTRY$CODE = 0;
NETWORK$CODE = 0;
HOST$CODE = 0;
DESTINATION$ADDRESS = 0;
SOURCE$ADDRESS = 0;
DESTINATION = 8;

CONTROL$CODE = LCOXTB(16) AND OFH;
IF CONTROL$CODE = 00 THEN
DO;
  COUNTRY$CODE = (ROR(LCOXTB(16), 4) AND OFH);
  IF COUNTRY$CODE = THIS$COUNTRY$CODE THEN
DO;
  NETWORK$CODE = LCOXTB(17) AND OFH;
  IF (NETWORK$CODE <= MAX$NETWORK$CODE) THEN
DO;
  IF (COUNTRY$CODE <> THIS$COUNTRY$CODE) OR
    ((NETWORK$CODE) <> THIS$UNID$NBR) THEN
    DESTINATION = 6; /* LOC TO NET */
  ELSE DESTINATION = 5; /* LOC TO LOC */
  LOBITS = (ROR(LCOXTB(17), 4) AND OFH);
  HIBITS = (ROL(LCOXTB(18), 4) AND OFOH);
  HOST$CODE = LOBITS OR HIBITS;
  IF (HOST$CODE >= 0) AND (HOST$CODE <= 63) THEN
    DESTINATION$ADDRESS = NETWORK$CODE OR 10H;
  ELSE
    IF (HOST$CODE >= 64) AND (HOST$CODE <= 127) THEN

```

```

SOURCE CODE FOR SIM544.SRC,          30 SEP 85

      DESTINATION$ADDRESS = NETWORK$CODE OR 20H;
    ELSE
      IF (HOST$CODE >= 128) AND (HOST$CODE <= 191) THEN
        DESTINATION$ADDRESS = NETWORK$CODE OR 30H;
      ELSE
        IF (HOST$CODE >= 192) AND (HOST$CODE <= 255) THEN
          DESTINATION$ADDRESS = NETWORK$CODE OR 40H;
        END;
      ELSE
        DO:
          CALL SNDSEQ(.TP$53A, SIZE(TP$53A));
          STATTB(04) = STATTB(04) + 1;
          STATTB(00) = STATTB(00) + 1;
        END;
      ELSE
        DO:
          /* NOT FOR THIS COUNTRY, SEND TO NETWORK/UNID 0 */
          CALL SNDSEQ(.TP$53B, SIZE(TP$53B));
          STATTB(03) = STATTB(03) + 1;
          STATTB(00) = STATTB(00) + 1;
        END;
      END;
    ELSE
      DO:
        /* IT IS AT THIS POINT THAT OTHER IP CONTROL CODES WILL
           BE INCORPORATED INTO THE NETWORK */
        CALL SNDSEQ(.TP$53C, SIZE(TP$53C));
        STATTB(02) = STATTB(02) + 1;
        STATTB(00) = STATTB(00) + 1;
      END;
    END;
  ELSE
    DO:
      CALL SNDSEQ(.TP$51B, SIZE(TP$51B));
      CALL HEX$ASC(CONTROL$CODE, 0);
      CALL SNDSEQ(.TEMP1, 2);
      CALL SNDSEQ(.TP$51C, SIZE(TP$51C));
      CALL HEX$ASC(COUNTRY$CODE, 0);
      CALL SNDSEQ(.TEMP1, 2);
      CALL SNDSEQ(.TP$51A, SIZE(TP$51A));
      CALL HEX$ASC(NETWORK$CODE, 0);
      CALL SNDSEQ(.TEMP1, 2);
      CALL SNDSEQ(.TP$51D, SIZE(TP$51D));
      CALL HEX$ASC(LOBITS, 0);
      CALL SNDSEQ(.TEMP1, 2);
      CALL SNDSEQ(.TP$51E, SIZE(TP$51E));
      CALL HEX$ASC(HIBITS, 0);
      CALL SNDSEQ(.TEMP1, 2);
      CALL SNDSEQ(.TP$52A, SIZE(TP$52A));
      CALL HEX$ASC(HOST$CODE, 0);
      CALL SNDSEQ(.TEMP1, 2);
      CALL SNDSEQ(.TP$52, SIZE(TP$52));
    END;
  END;

```

SOURCE CODE FOR SIM544.SRC, 30 SEP 85

```

CALL HEX$ASC(DESTINATION$ADDRESS, 0);
CALL SNDSEQ(.TEMP1, 2);

IF DESTINATION = 6 THEN /* LOC TO NET, GET SOURCE INFO */
DO;
SRC$NET$CODE = LCOXTB(13) AND OFH;
IF (SRC$NET$CODE <= MAX$NETWORK$CODE) THEN
DO;
LOBITS = (ROR(LCOXTB(13), 4) AND OFH);
HIBITS = (ROL(LCOXTB(14), 4) AND OFH);
SRC$HOST$CODE = LOBITS OR HIBITS;
IF (SRC$HOST$CODE >= 0) AND (SRC$HOST$CODE <= 63) THEN
SOURCE$ADDRESS = SRC$NET$CODE OR 10H;
ELSE
IF (SRC$HOST$CODE >= 64) AND (SRC$HOST$CODE <= 127) THEN
SOURCE$ADDRESS = SRC$NET$CODE OR 20H;
ELSE
IF (SRC$HOST$CODE >= 128) AND (SRC$HOST$CODE <= 191) THEN
SOURCE$ADDRESS = SRC$NET$CODE OR 30H;
ELSE
IF (SRC$HOST$CODE >= 192) AND (SRC$HOST$CODE <= 255) THEN
CALL SNDSEQ(.TP$52B, SIZE(TP$52B));
CALL HEX$ASC(SOURCE$ADDRESS, 0);
CALL SNDSEQ(.TEMP1, 2);
END;
ELSE DO;
DESTINATION = 8;
CALL SNDSEQ(.TP$53AA, SIZE(TP$53AA));
STATTB(04) = STATTB(04) + 1;
STATTB(00) = STATTB(00) + 1;
END;
END;

END;

END DET$ADDR;

/*****
* DATE: 2 OCT 84
* VERSION: 1.1
* NAME: DET$ADDR$NL
* MODULE NUMBER: 6.9
* DESCRIPTION: This module determines the destination from the second byte of the packet header.
* PASSED VARIABLES: None
* RETURNS: None
* GLOBAL VARIABLES USED: PORT, STATTB, NTLCLS
* GLOBAL VARIABLES CHANGED: PORT, STATTB
* MODULES CALLED: SENDSEQ
*****/

```

```

SOURCE CODE FOR SIM544.SRC.                30 SEP 85

* CALLING MODULES:                          ROUTE$IN
* AUTHOR: C. T. Childress
* HISTORY: 1.1 C.T. Childress - 2 OCT 84 - original SBC 544
*          implementation.
*          *****/
DETSADDR$NL: PROCEDURE BYTE ;

  DECLARE PORT BYTE,
           BITS BYTE;

  BITS = (ROR(NTLCB(NTLCNS), 4) AND OFH);
  IF (BITS >= 1 AND BITS <= 4) THEN
    DO CASE BITS:
      PORT = 1;
      PORT = 2;
      PORT = 3;
      PORT = 4;
    END;
  ELSE
    DO;
      PORT = 8;
      STATTB(01) = STATTB(01) + 1; /* INCREMENT LOCAL ERROR COUNT */
    END;
  RETURN PORT;
END DETSADDR$NL;

/*****
* DATE: 2 OCT 84
* VERSION: 1.1
* NAME: MOVESLL
* MODULE NUMBER: 6.3
* DESCRIPTION: This module routes packets between
               the local host tables. The datagrams are not formatted
               as packets, but moved as datagrams. The procedure
               moves a DATAGRAM$SIZE segment starting at the location
               indicated by TABLE$PTR to the location of the next
               available byte in the designated port.
* PASSED VARIABLES: TABLE$PTR, PORT
* RETURNS: None
* GLOBAL VARIABLES USED: TX01NE, TX02NE, TX03NE, TX04NE, STATTB,
                          L$RO$DEST$ERR
* GLOBAL VARIABLES CHANGED: STATTB
* MODULES CALLED: LOSTAB$H$SKP, SND$SEQ
* CALLING MODULES: ROUTE$IN
* AUTHOR: C. T. Childress
* HISTORY:
*****/

```

```

*      1.1 C.T. Childress - 2 OCT 84 - original SBC 544      *
*      implementation.                                       *
*****
MOVE$LL: PROCEDURE(TABLE$PTR, PORT);

```

```

    DECLARE PORT BYTE,
    TABLE$PTR ADDRESS,
    LCOXTB BASED TABLE$PTR(1) BYTE;

```

```

    IF(PORT >= 1 AND PORT <= 4) THEN
    DO CASE PORT;

```

```

        ;      /* CASE ZERO IS NULL */
        DO;      /* CASE ONE */
            IF TRANS$1$RDY = TRUE THEN DO;
                TRANS$1$RDY = FALSE;
                CALL SNDSEQ(.TP$24, SIZE(TP$24));
                CALL MOVE(DATA$GRAM$SIZE, TABLE$PTR, .TX01TB(TX01NE));
                CALL LD$TAB$H$SKP(8);
                CALL TRNMIT$PKT(PORT);
            END;
        END;

```

```

        /* CASE TWO */
        DO;      IF TRANS$2$RDY = TRUE THEN DO;
            TRANS$2$RDY = FALSE;
            CALL SNDSEQ(.TP$25, SIZE(TP$25));
            CALL MOVE(DATA$GRAM$SIZE, TABLE$PTR, .TX02TB(TX02NE));
            CALL LD$TAB$H$SKP(9);
            CALL TRNMIT$PKT(PORT);
        END;

```

```

        /* CASE THREE */
        DO;      IF TRANS$3$RDY = TRUE THEN DO;
            TRANS$3$RDY = FALSE;
            CALL SNDSEQ(.TP$26, SIZE(TP$26));
            CALL MOVE(DATA$GRAM$SIZE, TABLE$PTR, .TX03TB(TX03NE));
            CALL LD$TAB$H$SKP(10);
            CALL TRNMIT$PKT(PORT);
        END;

```

```

        /* CASE FOUR */
        DO;      IF TRANS$4$RDY = TRUE THEN DO;
            TRANS$4$RDY = FALSE;
            CALL SNDSEQ(.TP$27, SIZE(TP$27));
            CALL MOVE(DATA$GRAM$SIZE, TABLE$PTR, .TX04TB(TX04NE));
            CALL LD$TAB$H$SKP(11);

```


SOURCE CODE FOR SIM544.SRC. 30 SEP 85

```

/*          CALL TRNMITS$PKT(PORT);
*/
END;

END;

/* END CASE */
ELSE DO;
  CALL SNDSEQ(.TP$28, SIZE(TP$28));
  STATTB(L$ROS$DESTERR) = STATTB(L$ROS$DESTERR) + 1;
END;

END MOVE$LL;

/*****
* DATE: 2 OCT 84
* VERSION: 1.1
* NAME: MOVE$NL
* MODULE NUMBER: 6.10
* DESCRIPTION: This module routes packets between
*               the local host tables and the NTLCTB table.
*               The destination address is an input parameter telling
*               which host(port) the packet should be routed to.
* PASSED VARIABLES: PORT
* RETURNS: None
* GLOBAL VARIABLES USED: NTLCTB, TX01NE (2, 3, 4),
*                          TX01TB( 2, 3, 4)
* GLOBAL VARIABLES CHANGED: TX01TB(2, 3, 4)
* MODULES CALLED: MOVE$LL, SND$SEQ, BUILD$I$PACKET,
*                  LD$TAB$H$SKP
* CALLING MODULES: ROUTE$IN
* AUTHOR: C. T. Childress
* HISTORY: 1.1 C.T. Childress - 2 OCT 84 - original SBC 544
*               implementation.
*****/

MOVE$NL: PROCEDURE(PORT);
  DECLARE PORT BYTE;

  IF (PORT >= 1 AND PORT <= 4) THEN
    DO CASE PORT;
      ;
    DO;
      CALL MOVE(DATA$GRAM$SIZE, .NTLCTB(NTLCNS + 5), .TX01TB(TX01NE));
      CALL SNDSEQ(.TP$30, SIZE(TP$30));
      CALL LD$TAB$H$SKP(8);
    END;
  DO;

```

```

SOURCE CODE FOR SIM544.SRC,          30 SEP 85

      CALL MOVE(DATA$GRAM$SIZE, .NTLCTB(NTLCLS + 5), .TX02TB(TX02NE));
      CALL SNDSEQ(.TP$31, SIZE(TP$31));
      CALL LD$TAB$H$SKP(9);
      END;

DO;
      CALL MOVE(DATA$GRAM$SIZE, .NTLCTB(NTLCLS + 5), .TX03TB(TX03NE));
      CALL SNDSEQ(.TP$32, SIZE(TP$32));
      CALL LD$TAB$H$SKP(10);
      END;

DO;
      CALL MOVE(DATA$GRAM$SIZE, .NTLCTB(NTLCLS + 5), .TX04TB(TX04NE));
      CALL SNDSEQ(.TP$33, SIZE(TP$33));
      CALL LD$TAB$H$SKP(11);
      END;

      END;
ELSE
      CALL SNDSEQ(.TP$34, SIZE(TP$34));

```

```

END MOVE$NL;

/*****
* DATE:          2 OCT 84
* VERSION:       1.1
* NAME:          ROUTESIN
* MODULE NUMBER: 6.0
* DESCRIPTION:   This module routes datagrams from the
*               local receive tables to local transmit tables, if
*               destined for another local host, or to the datalink
*               layer if destined for another UNID. The address byte
*               is evaluated for the destination processing. This
*               procedure along with ROUTE$OUT is called in and endless
*               loop by the main program.
* PASSED VARIABLES:
* RETURNS:       None
* GLOBAL VARIABLES USED:  DESTINATION, STATTB, LCO1TB(2, 3, 4),
*                           DESTINATION$ADDRESS
* GLOBAL VARIABLES CHANGED:  All of those above
* MODULES CALLED:  SENDSEQ, SRVC$TAB$H$SKP, DET$ADDR,
*                   MOVE$LL, SEND$I$PACKET, DET$ADDR$NL, MOVE$NL HEX$ASC
* CALLING MODULES:
* AUTHOR:        C. T. Childress
* HISTORY:       1.1 C.T. Childress - 2 OCT 84 - original SBC 544
*               implementation.
*****/

```

30 SEP 85

SOURCE CODE FOR SIM544.SRC,

ROUTE\$IN: PROCEDURE;

```

IF(((LC01NE - LC01NS) >= DATA$GRAM$SIZE) OR (LC01NS > LC01NE)) THEN
DO:
CALL SNDSEQ(.TP$6, SIZE(TP$6));
CALL DET$ADDR(.LC01TB(LC01NS));
IF DESTINATION = 5 THEN
DO:
CALL SNDSEQ(.TP$7, SIZE(TP$7));
CALL MOVE$LL(.LC01TB(LC01NS), (ROR(DESTINATION$ADDRESS, 4) AND 0FH));
END;
ELSE
IF DESTINATION = 6 THEN
DO:
CALL SNDSEQ(.TP$8, SIZE(TP$8));
CALL BUILD$I$PACKET(.LC01TB(LC01NS));
END;
ELSE
DO:
CALL SNDSEQ(.TP$9, SIZE(TP$9));
STATTB(L$RI$DEST$ERR) = STATTB(L$RI$DEST$ERR) + 1;
END;
CALL SRVC$TAB$H$SKP(1);
END;
IF(((LC02NE - LC02NS) >= DATA$GRAM$SIZE) OR (LC02NS > LC02NE)) THEN
DO:
CALL SNDSEQ(.TP$10, SIZE(TP$10));
CALL DET$ADDR(.LC02TB(LC02NS));
IF DESTINATION = 5 THEN
DO:
CALL SNDSEQ(.TP$11, SIZE(TP$11));
CALL MOVE$LL(.LC02TB(LC02NS), (ROR(DESTINATION$ADDRESS, 4) AND 0FH));
END;
ELSE
IF DESTINATION = 6 THEN
DO:
CALL SNDSEQ(.TP$12, SIZE(TP$12));
CALL BUILD$I$PACKET(.LC02TB(LC02NS));
END;
ELSE
DO:
CALL SNDSEQ(.TP$13, SIZE(TP$13));
CALL SNDSEQ(.TP$51, SIZE(TP$51));
CALL HEX$ASC(LOW(DESTINATION), 0);
CALL SNDSEQ(.TEMP1, 2);
CALL SNDSEQ(.TP$52, SIZE(TP$52));
CALL HEX$ASC(DESTINATION$ADDRESS, 0);
CALL SNDSEQ(.TEMP1, 2);
STATTB(L$RI$DEST$ERR) = STATTB(L$RI$DEST$ERR) + 1;

```

SOURCE CODE FOR SIM544.SRC. 30 SEP 85

```

END;
CALL SRVC$TAB$H$SKP(2);
END;

IF(((LC03NE - LC03NS) >= DATA$GRAM$SIZE) OR (LC03NS > LC03NE)) THEN
DO;
CALL SNDSEQ(.TP$14, SIZE(TP$14));
CALL DET$ADDR(.LC03TB(LC03NS));
IF DESTINATION = 5 THEN
DO;
CALL SNDSEQ(.TP$15, SIZE(TP$15));
CALL MOVE$LL(.LC03TB(LC03NS), (ROR(DESTINATION$ADDRESS, 4) AND OFH));
END;
ELSE
IF DESTINATION = 6 THEN
DO;
CALL SNDSEQ(.TP$16, SIZE(TP$16));
CALL BUILD$I$PACKET(.LC03TB(LC03NS));
END;
ELSE
DO;
CALL SNDSEQ(.TP$17, SIZE(TP$17));
STATTB(L$R1$DEST$ERR) = STATTB(L$R1$DEST$ERR) + 1;
END;
CALL SRVC$TAB$H$SKP(3);
END;

IF(((LC04NE - LC04NS) >= DATA$GRAM$SIZE) OR (LC04NS > LC04NE)) THEN
DO;
CALL SNDSEQ(.TP$18, SIZE(TP$18));
CALL DET$ADDR(.LC04TB(LC04NS));
IF DESTINATION = 5 THEN
DO;
CALL SNDSEQ(.TP$19, SIZE(TP$19));
CALL MOVE$LL(.LC04TB(LC04NS), (ROR(DESTINATION$ADDRESS, 4) AND OFH));
END;
ELSE
IF DESTINATION = 6 THEN
DO;
CALL SNDSEQ(.TP$20, SIZE(TP$20));
CALL BUILD$I$PACKET(.LC04TB(LC04NS));
END;
ELSE
DO;
CALL SNDSEQ(.TP$21, SIZE(TP$21));
STATTB(L$R1$DEST$ERR) = STATTB(L$R1$DEST$ERR) + 1;
END;
CALL SRVC$TAB$H$SKP(4);
END;

```

```

SOURCE CODE FOR SIM544.SRC,          30 SEP 85

IF ((NTLCNE - NTLNS) >= PACKET$SIZE) OR (NTLCNS > NTLCTB) THEN
DO:
  DESTINATION$ADDRESS = DET$ADDR$NL;
  CALL MOVES$NL(DESTINATION$ADDRESS);
  CALL SRVCTAB$H$SKP(7);
END;

```

```

END ROUTES$IN;

```

```

/*****
* DATE:          18 SEP 85
* VERSION:       1.2
* NAME:          SERVICES$LOOP
* MODULE NUMBER: 7.0
* DESCRIPTION:    This module routes a frame back to the
*                network layer software. The source and destination
*                addresses are switched.
* PASSED VARIABLES:  NONE
* RETURNS:          None
* GLOBAL VARIABLES USED:  NTLCTB, NTLCTNE,
*                GLOBAL VARIABLES CHANGED:  All of those above
*                MODULES CALLED:  LD$TAB$H$SKP, SRVCTAB$H$SKP
*                CALLING MODULES:  LOOP
* AUTHOR:         Mark Weber
* HISTORY:        1.2 Mark Weber - 18 SEP 85 - changed to X.25 packet format
*                1.1 C.T. Childress - 2 OCT 84 - original SBC 544
*                implementation.
*****
SERVICES$LOOP: PROCEDURE;
  DECLARE INDEX ADDRESS;

  IF ((LCNTNE - LCNTNS) >= PACKET$SIZE) OR
     (LCNTNS > LCNTNE) THEN
DO:
  CALL SNDSEQ(.TP$40, SIZE(TP$40));
  DO INDEX = 0 TO (PACKET$SIZE - 1);
    NTLCTB(NTLCNE + INDEX) = LCNTTB(LCNTNS + INDEX);
  END;
  NTLCTB(NTLCNE + 4) = LCNTTB(LCNTNS + 5);
  NTLCTB(NTLCNE + 5) = LCNTTB(LCNTNS + 4); /* SWAP PACKET HEADER */
  DO INDEX = 0 TO 3;
    NTLCTB(NTLCNE + INDEX + 22) = LCNTTB(LCNTNS + INDEX + 26); /* SWAP IP DEST & SOURCE */
    NTLCTB(NTLCNE + INDEX + 26) = LCNTTB(LCNTNS + INDEX + 22);
  END;
  CALL LD$TAB$H$SKP(7);
  CALL SRVCTAB$H$SKP(6);
END;

END SERVICES$LOOP;

```

SOURCE CODE FOR SIM544.SRC. 30 SEP 85

```

*****
* DATE: 2 OCT 84 *****
* VERSION: 1.1 *****
* NAME: LOAD *****
* MODULE NUMBER: 4.5 *****
* DESCRIPTION: This module routes datagrams from the *****
* local transmit tables to the local hosts. Version 1.2 *****
* implements the X.25 packet format *****
* CHAN$PTR, DEST$HOST, DEST$UNID *****
* PASSED VARIABLES: *****
* RETURNS: None *****
* GLOBAL VARIABLES USED: MSGUM *****
* GLOBAL VARIABLES CHANGED: LCO1TB (2, 3, 4) *****
* MODULES CALLED: SENDSEQ *****
* CALLING MODULES: READ$LINE *****
* AUTHOR: C. T. CHILDRESS *****
* HISTORY: 1.1 C.T. Childress - 2 OCT 84 - original SBC 544 *****
* implementation. *****
*****
LOAD: PROCEDURE(CHAN$PTR, DEST$UNID, DEST$HOST);
*****
DECLARE (INDEX, CHAN$PTR) ADDRESS;
DECLARE (DEST$UNID, DEST$HOST) BYTE;
DECLARE LCOXTB BASED CHAN$PTR(1) BYTE;

DO INDEX = 0 TO (DATA$GRAM$SIZE - 1);
  LCOXTB(INDEX) = '*';
END;

/* IP SOURCE HEADER */
LCOXTB( 12) = 10H; /* CC = 1, CT = 0 */
LCOXTB( 13) = (THIS$UNID$NBR AND OFH) OR
  (ROL(LOW(CHAN$NUM*56),4) AND OFOH);
/* HC(L) = 6, NC = 2 */
LCOXTB( 14) = 70H OR (ROR(LOW(CHAN$NUM*56),4) AND OFH);
/* PC(0) = 7, HC(H) = 0 */
LCOXTB( 15) = 00H; /* PC(2) = 0, PC(1) = 0 */

/* IP DESTINATION HEADER */
LCOXTB( 16) = 10H; /* CC = 1, CT = 0 */
LCOXTB( 17) = (DEST$UNID AND OFH) OR
  ROL((DEST$HOST AND OFH), 4);
/* HC(L) = 6, NC = 2 */
LCOXTB( 18) = ROR((DEST$HOST AND OFH),4) OR 70H;
/* PC(0) = 7, HC(H) = 0 */
LCOXTB( 19) = 00H; /* PC(2) = 0, PC(1) = 0 */

CALL MOVE(TCP$DATA$SIZE, .MESSAGE, .LCOXTB( 56));
IF MSGNUM > 9 THEN
  MSGNUM = 0;

```

SOURCE CODE FOR SIMS44.SRC, 30 SEP 85

```

LCOXTB( 86) = '0' + MSGNUM;
MSGNUM = MSGNUM + 1;
CALL SNDSEQ(.TP$54, SIZE(TP$54));

```

END LOAD;

```

/*****
* DATE: 25 SEP 84
* VERSION: 1.0
* NAME: READ$TX$TAB:
* MODULE NUMBER: 8.0
* FUNCTION: Reads the local transmit table
* INPUTS: None
* OUTPUTS: None
* GLOBAL VARIABLES USED: TX01TB (2, 3, 4), TX01TS(2, 3, 4), TEMP2
* GLOBAL VARIABLES CHANGED: all of those above
* MODULES CALLED: SRVC$TAB$H$SKP, HEX$ASC, SNDSEQ
* CALLING MODULES: main
* AUTHOR: Capt C.T. Childress
* HISTORY:
* 1.0 - Capt C. T. Childress
*****/

```

```

READ$TX$TAB: PROCEDURE;
DECLARE I BYTE;

```

```

IF ((TX01NE - TX01NS) >= DATA$GRAM$SIZE) OR (TX01NS > TX01NE) THEN

```

```

DO:
  I = 12;
DO WHILE I <= 18;
  CALL HEX$ASC(TX01TB(TX01NS + I), (I-12)*2);
  I = I + 1;
END;

```

```

I = 0;
DO WHILE I <= 15;
  TEMP2(I) = TEMP1(14-I);
  TEMP2(I+1) = TEMP1(15-I);
  I = I + 2;
END;

```

```

CALL SNDSEQ(.TP$55A, SIZE(TP$55A));
CALL SNDSEQ(.CRLF, SIZE(CRLF));
CALL SNDSEQ(.TEMP2, SIZE(TEMP2));
CALL SNDSEQ(.TX01TB(TX01NS + 56), TCP$DATA$SIZE);
CALL SRVC$TAB$H$SKP(8);

```

END;

```

IF ((TX02NE - TX02NS) >= DATA$GRAM$SIZE) OR (TX02NS > TX02NE) THEN

```

```

DO:
  I = 12;
DO WHILE I <= 18;

```

```

      CALL HEX$ASC(TX02TB(TX02NS + I),(I-12)*2);
      I = I + 1;
    END;
    I = 0;
    DO WHILE I <= 15;
      TEMP2(I) = TEMP1(14-I);
      TEMP2(I+1) = TEMP1(15-I);
      I = I + 2;
    END;
    CALL SNDSEQ(.TP$55B, SIZE(TP$55B));
    CALL SNDSEQ(.CRLF, SIZE(CRLF));
    CALL SNDSEQ(.TEMP2, SIZE(TEMP2));
    CALL SNDSEQ(.TX02TB(TX02NS + 56), TCP$DATA$SIZE);
    CALL SRVC$TAB$H$SKP(9);
  END;

  IF ((TX03NE - TX03NS) >= DATA$GRAM$SIZE) OR (TX03NS > TX03NE) THEN
  DO;
    I = 12;
    DO WHILE I <= 18;
      CALL HEX$ASC(TX03TB(TX03NS + I),(I-12)*2);
      I = I + 1;
    END;
    I = 0;
    DO WHILE I <= 15;
      TEMP2(I) = TEMP1(14-I);
      TEMP2(I+1) = TEMP1(15-I);
      I = I + 2;
    END;
    CALL SNDSEQ(.TP$55C, SIZE(TP$55C));
    CALL SNDSEQ(.CRLF, SIZE(CRLF));
    CALL SNDSEQ(.TEMP2, SIZE(TEMP2));
    CALL SNDSEQ(.TX03TB(TX03NS + 56), TCP$DATA$SIZE);
    CALL SRVC$TAB$H$SKP(10);
  END;

  IF ((TX04NE - TX04NS) >= DATA$GRAM$SIZE) OR (TX04NS > TX04NE) THEN
  DO;
    I = 12;
    DO WHILE I <= 18;
      CALL HEX$ASC(TX04TB(TX04NS + I),(I-12)*2);
      I = I + 1;
    END;
    I = 0;
    DO WHILE I <= 15;
      TEMP2(I) = TEMP1(14-I);
      TEMP2(I+1) = TEMP1(15-I);
      I = I + 2;
    END;
    CALL SNDSEQ(.TP$55D, SIZE(TP$55D));
  END;

```


SOURCE CODE FOR SIM544.SRC, 30 SEP 85

```

      CALL SNDSEQ(.CRLF, SIZE(CRLF));
      CALL SNDSEQ(.TEMP2, SIZE(TEMP2));
      CALL SNDSEQ(.TX04TB(TX04NS + 56), TCP$DATA$SIZE);
      CALL SRVC$TAB$H$SKP(11);
END;

```

```

END READ$XTAB;

```

```

/*****
* DATE: 25 SEP 84
* VERSION: 1.0
* NAME: READ$XTAB
* MODULE NUMBER: 5.0
* FUNCTION: Reads the local receive tables
* INPUTS: None
* OUTPUTS: None
* GLOBAL VARIABLES USED: LCO1TB (2, 3, 4), LCO1NS (2, 3, 4), TEMP2
* GLOBAL VARIABLES CHANGED: LCO1TB (2, 3, 4), TEMP2
* MODULES CALLED: HEX$ASC, SNDSEQ, SRVC$TAB$H$SKP
* CALLING MODULES: main
* AUTHOR: Capt C.T. Childress
* HISTORY: 1.0 - Capt C. T. Childress
*****/
READ$XTAB: PROCEDURE;
DECLARE I BYTE;

```

```

      IF ((LCO1NE - LCO1NS) >= DATA$GRAM$SIZE) OR (LCO1NS > LCO1NE) THEN

```

```

DO;

```

```

  I = 12;

```

```

  DO WHILE I <= 18;

```

```

    CALL HEX$ASC(LCO1TB(LCO1NS + I), (I-12)*2);

```

```

    I = I + 1;

```

```

  END;

```

```

  I = 0;

```

```

  DO WHILE I <= 15;

```

```

    TEMP2(I) = TEMP1(14-I);

```

```

    TEMP2(I+1) = TEMP1(15-I);

```

```

    I = I + 2;

```

```

  END;

```

```

  CALL SNDSEQ(.TP$56A, SIZE(TP$56A));

```

```

  CALL SNDSEQ(.CRLF, SIZE(CRLF));

```

```

  CALL SNDSEQ(.TEMP2, SIZE(TEMP2));

```

```

  CALL SNDSEQ(.LCO1TB(LCO1NS + 56), TCP$DATA$SIZE);

```

```

END;

```

```

      IF ((LCO2NE - LCO2NS) >= DATA$GRAM$SIZE) OR (LCO2NS > LCO2NE) THEN

```

```

DO;

```

```

  I = 12;

```

```

  DO WHILE I <= 18;

```

```

      CALL HEX$ASC(LC02TB(LC02NS + I),(I-12)*2);
      I = I + 1;
    END;
    I = 0;
    DO WHILE I <= 15;
      TEMP2(I) = TEMP1(14-I);
      TEMP2(I+1) = TEMP1(15-I);
      I = I + 2;
    END;
    CALL SNDSEQ(.TP$56B, SIZE(TP$56B));
    CALL SNDSEQ(.CRLF, SIZE(CRLF));
    CALL SNDSEQ(.TEMP2, SIZE(TEMP2));
    CALL SNDSEQ(.LC02TB(LC02NS + 56), TCP$DATA$SIZE);
  END;

  IF ((LC03NE - LC03NS) >= DATA$GRAM$SIZE) OR (LC03NS > LC03NE) THEN
    DO;
      I = 12;
      DO WHILE I <= 18;
        CALL HEX$ASC(LC03TB(LC03NS + I),(I-12)*2);
        I = I + 1;
      END;
      I = 0;
      DO WHILE I <= 15;
        TEMP2(I) = TEMP1(14-I);
        TEMP2(I+1) = TEMP1(15-I);
        I = I + 2;
      END;
      CALL SNDSEQ(.TP$56C, SIZE(TP$56C));
      CALL SNDSEQ(.CRLF, SIZE(CRLF));
      CALL SNDSEQ(.TEMP2, SIZE(TEMP2));
      CALL SNDSEQ(.LC03TB(LC03NS + 56), TCP$DATA$SIZE);
    END;

    IF ((LC04NE - LC04NS) >= DATA$GRAM$SIZE) OR (LC04NS > LC04NE) THEN
      DO;
        I = 12;
        DO WHILE I <= 18;
          CALL HEX$ASC(LC04TB(LC04NS + I),(I-12)*2);
          I = I + 1;
        END;
        I = 0;
        DO WHILE I <= 15;
          TEMP2(I) = TEMP1(14-I);
          TEMP2(I+1) = TEMP1(15-I);
          I = I + 2;
        END;
        CALL SNDSEQ(.TP$56D, SIZE(TP$56D));
        CALL SNDSEQ(.CRLF, SIZE(CRLF));
        CALL SNDSEQ(.TEMP2, SIZE(TEMP2));
      END;

```

SOURCE CODE FOR SIM544.SRC, 30 SEP 85
 CALL SNDSEQ(.LC04TB(LC04NS + 56), TCP\$DATA\$SIZE);

END;

END READ\$LCPTAB;

```

/*****
* DATE: 20 SEP 84
* VERSION: 1.0
* NAME: UPDATE$LCPTAB
* MODULE NUMBER: 4.7
* FUNCTION: Sets the channel pointer to the current position
* of a table passed to the procedure.
* INPUTS: CHAN$NUM
* OUTPUTS: CHAN$NUM
* GLOBAL VARIABLES USED: LC01TB (2, 3, 4), LC01NE (2, 3, 4)
* GLOBAL VARIABLES CHANGED: LC01TB (2, 3, 4), LC01NE (2, 3, 4)
* MODULES CALLED: None
* CALLING MODULES: READLINE
* AUTHOR: C. T. Childress
* HISTORY: 1.0 - Capt C. T. Childress
*****/
UPDATE$LCPTR: PROCEDURE(CHAN$NUM) ADDRESS;
DECLARE (CHAN$NUM, CHAN$PTR) ADDRESS;

```

```

IF (CHAN$NUM >= 1 AND CHAN$NUM <= 4) THEN
  DO CASE CHAN$NUM;

```

```

    CHAN$PTR = .LC01TB(LC01NE);
    CHAN$PTR = .LC02TB(LC02NE);
    CHAN$PTR = .LC03TB(LC03NE);
    CHAN$PTR = .LC04TB(LC04NE);
  END;

```

RETURN CHAN\$PTR;

END UPDATE\$LCPTR;

```

/*****
* DATE: 20 SEP 84
* VERSION: 1.0
* NAME: READ$LINE
* MODULE NUMBER: 4.0
* FUNCTION: Reads a line of data from the host console and
* then loads the specified number of datagrams into the
* transmit table according to the specified path
* INPUTS: None
* OUTPUTS: None
* GLOBAL VARIABLES USED: BUFFER, CHAN$NUM
* GLOBAL VARIABLES CHANGED: BUFFER
*****/

```

```

SOURCE CODE FOR SIM544.SRC,          30 SEP 85

* MODULES CALLED:          SNDSEQ, ERR$CHK, READ, ASC$HEX, HEX$ASC, *
* LD$TAB$TAB              main
* CALLING MODULES:
* AUTHOR: C. T. Childress
* HISTORY:
*          1.0 - Capt C. T. Childress
*          *****/
READ$LINE: PROCEDURE;

DECLARE CHAN$PTR ADDRESS,
        (I, DEST$NET$CODE, DEST$HOST$CODE) BYTE;

I = 0;

CALL SNDSEQ(.MSG1, SIZE(MSG1));
CALL READ(R$CONN, .BUFFER, 128, .ACTUAL, .STATUS);
CALL ERR$CHK;

IF (BUFFER(0) = 'Y') OR (BUFFER(0) = 'y') THEN
DO;
    CALL SNDSEQ(.MSG3, SIZE(MSG3));
    CALL READ(R$CONN, .BUFFER, 128, .ACTUAL, .STATUS);
    CALL ERR$CHK;
    CHAN$NUM = DOUBLE(ASC$HEX(BUFFER(0)));
    IF (CHAN$NUM >= 1) AND (CHAN$NUM <= 4) THEN
    DO;
        CHAN$PTR = UPDATE$LCPTR(CHAN$NUM);
        CALL SNDSEQ(.MSG5, SIZE(MSG5));
        CALL READ(R$CONN, .BUFFER, 128, .ACTUAL, .STATUS);
        CALL ERR$CHK;
        DEST$NET$CODE = ASC$HEX(BUFFER(0));

        CALL SNDSEQ(.MSG6, SIZE(MSG6));
        CALL READ(R$CONN, .BUFFER, 128, .ACTUAL, .STATUS);
        CALL ERR$CHK;
        DEST$HOST$CODE = 0;
        DO I = 1 TO (ACTUAL-2);
            DEST$HOST$CODE = ROL(DEST$HOST$CODE, 4);
            DEST$HOST$CODE = DEST$HOST$CODE OR ASC$HEX(BUFFER(I-1));
        END;

        CALL SNDSEQ(.MSG4, SIZE(MSG4));
        CALL READ(R$CONN, .BUFFER, 128, .ACTUAL, .STATUS);
        CALL ERR$CHK;
        IF (BUFFER(0) >= '1') AND (BUFFER(0) <= '9') THEN
        DO I = 1 TO ASC$HEX(BUFFER(0));
            CALL LOAD(CHAN$PTR, DEST$NET$CODE, DEST$HOST$CODE);
            CALL LD$TAB$SKP(CHAN$NUM);
            CHAN$PTR = UPDATE$LCPTR(CHAN$NUM);
        END;
    END;

```

SOURCE CODE FOR SIMS44.SRC, 30 SEP 85

```

END;
END;

CALL SNDSEQ(.TP$50, SIZE(TP$50));
CALL HEX$ASC(LOW(CHAN$NUM), 0);
CALL SNDSEQ(.TEMP1, 2);
CALL SNDSEQ(.TP$51A, SIZE(TP$51A));
CALL HEX$ASC(DEST$NET$CODE, 0);
CALL SNDSEQ(.TEMP1, 2);
CALL SNDSEQ(.TP$52A, SIZE(TP$52A));
CALL HEX$ASC(DEST$HOST$CODE, 0);
CALL SNDSEQ(.TEMP1, 2);

CALL SNDSEQ(.MSG2, SIZE(MSG2));
CALL READ(R$CONN, .BUFFER, 128, .ACTUAL, .STATUS);
CALL ERR$CHK;

IF (BUFFER(0) = 'Y') OR (BUFFER(0) = 'Y') THEN
  FOREVER = FALSE;
ELSE
  FOREVER = TRUE;

END READ$LINE;

/*****
/* THIS IS THE MAIN BODY OF THE PROGRAM
*****/

BEGIN:

CALL SNDSEQ(.STARTUP$HDR, SIZE(STARTUP$HDR));
CALL SNDSEQ(.TP$1, SIZE(TP$1));
CALL INIT$TAB;
CALL SNDSEQ(.TP$2, SIZE(TP$2));
CALL SNDSEQ(.TP$3, SIZE(TP$3));
CALL INVINT;
CALL SNDSEQ(.TP$4, SIZE(TP$4));
FOREVER = TRUE;
CALL READ$LINE;
DO WHILE FOREVER;
  CALL READ$LC$TAB;
  CALL SNDSEQ(.TP$5, SIZE(TP$5));
  CALL ROUTE$IN;
  CALL SERVICE$LOOP;
  CALL SNDSEQ(.TP$22, SIZE(TP$22));
/*  CALL READ$TX$TAB;
  CALL SNDSEQ(.TP$35, SIZE(TP$35));
  CALL READ$LINE;
END;
CALL SNDSEQ(.TP$36, SIZE(TP$36));

```

SOURCE CODE FOR SIMS44.SRC.

30 SEP 85

PAGE 40

CALL EXIT;

END MAIN;

/***** THE END *****/

APPENDIX M

SBC 544 Operational Software

OP544.SRC contains the software modified for polled transmit I/O. This software was successfully tested in the LSI-11 network. This software is embedded on the SBC 544. The UNID is assigned UNID number 2, Country Code 9 for operation in the LSI-11 network. Refer to the EE690 gateway software for information on the LSI-11 gateway nodes and to chapter VI for testing within the LSI-11 network.

25 SEP 85

SOURCE CODE FOR OP544.SRC.

\$TITLE('LOCAL NETWORK TEST PROGRAM, 25 SEP 1985')

\$INTVECTOR(4,20H)

```

/*****
* DATE: 24 SEP 85
* VERSION: 1.3
* TITLE: ISO layer 3B SBC 544 IMPLEMENTATION WITH X.25 PACKET FORMAT
* FILENAME: OP544A.SRC
* COORDINATOR: Capt Mark W. Weber
* PROJECT: UNID II
* OPERATING SYSTEM: 8080/8085 CPU (embedded software)
* LANGUAGE: PL/M 80
* USE: This file requires the filename INTR3.LOC include file for oper-
      ating the interrupt driven receive routines and the polled trans-
      mit routines. The include file INTR.LOC has both transmit and
      receive routines interrupt driven. This file must be linked with
      PLM80.LIB, SYSTEM.LIB. This file maybe linked and located with
      the submit command lmk544(sim544a).
* CONTENTS:
      MAIN - main processing routine
      INIT$N$TAB - initializes the necessary hardware
      INIT - initializes the 8255 jump table and vectors
      INIT$TAB - initializes the tables and table pointers
      HEX$ASC - converts HEX to ASCII
      ASC$HEX - converts ASCII to HEX
      VALID$HEX - validates HEX numbers
      SNDSEQ - writes messages to USART 0 for status updates
      LD$TAB$H$SKP - updates the transmit tables
      SRVC$TAB$H$SKP - updates the receive tables
      RXIN$1 - USART 0 (channel 1) monitor input routine
      TXOUT$1 - USART 0 transmit monitor routine
      SERVICE$RCV$1 - channel 1 receive interrupt routine
      SERVICE$RCV$2 - channel 2 receive interrupt routine
      SERVICE$RCV$3 - channel 3 receive interrupt routine
      SERVICE$RCV$4 - channel 4 receive interrupt routine
      SERVICE$TRANS$1 - channel 1 transmit pooled routine
      SERVICE$TRANS$2 - channel 2 transmit pooled routine
      SERVICE$TRANS$3 - channel 3 transmit pooled routine
      SERVICE$TRANS$4 - channel 4 transmit pooled routine
      SEND$PACKET - procedure formats a packet for the datalink layer
      DET$ADDR - procedure determines the destination of local data
      DET$ADDR$NL - determines the destination of data from the
                    datalink layer
      MOVE$TO$LOCAL - moves packets from the datalink layer to the
                    local network layer
      ROUTE$IN - processes incoming packets from the transport layer
      ROUTE$OUT - processes packets in the local network layer
                    transmit tables
      SERVICE$LOOP - software loop for routing datagrams back to the
                    local network layer
      LOOP - SBC 8B/45 semaphore simulation
*****/

```


25 SEP 85

SOURCE CODE FOR OP544.SRC.

```

*
* SET$TRTA - handles TRTA protocol
* main - calls procedures to simulate operational SBC 544 in UNID
*
* FUNCTION:
* Implements the operation of the SBC 544 in the UNID II using
* the TCP/IP protocol as defined by Phister. This SBC handles
* the functions of ISO layer 3B which processes the IP protocol
* and handles a multi-host requirements.
*
* HISTORY:
* 1.2 Mark Weber - implemented polled transmit routines
* 1.1 C.T. Childress - 2 OCT 84 - original implementation
* 1.0 P. Philster - original
*
*****
/* THE FOLLOWING ARE UNID DEFINED VARIABLES */
/* NOTE: THESE VARIABLES MAY CHANGE DEPENDING ON THE
SOFTWARE CONFIGURATION USED WITHIN THE DELNET */

DECLARE
  DATA$GRAM$SIZE LITERALLY '128', /* DATAGRAM FROM HOST */
  PACKET$SIZE LITERALLY '138', /* DATAGRAM + 5 FOR HEADER */
  FRAME$SIZE LITERALLY '140', /* PACKET + 2 FOR HEADER */
  PACKET$SIN$TABLE LITERALLY '10',
  STAT$NBR LITERALLY '20', /* STATUS ENTRIES IN STATB */
  DATA$TABLE$SIZE LITERALLY '1280', /* DATAGRAM * NBR OF DATAGRAMS */
  PACKET$TABLE$SIZE LITERALLY '1380', /* PACKET * NBR OF PACKETS */
  FRAME$TABLE$SIZE LITERALLY '1400', /* FRAME * NBR OF PACKETS */
  TCP$DATA$SIZE LITERALLY '72', /* TCP DATA SIZE */
  TR LITERALLY '42H', /* TRANSMIT REQUEST CHAR */
  TA LITERALLY '41H', /* TRANSMIT ACKNOWLEDGE CHAR */
  MAX$RXTA$TRIES LITERALLY '3', /* MAXIMUM NUMBER OF TA WAIT TRIES */
  OFFSET1 LITERALLY '10', /* OFFSET FROM PACKET TO IP HEADER */

/* FOLLOWING ARE NETWORK DEFINED VARIABLES */
/* NOTES: 1. THIS$UNID$NBR MUST REFLECT WHICH UNID THIS IS.
2. THIS$COUNTRY$CODE MUST REFLECT THE AREA TO WHICH
THIS UNID IS LOCATED.
3. MAX$COUNTRY$CODE WILL INDICATE WHICH COUNTRY CODES
ARE CURRENTLY OPERATIONAL. CC=0000 IS RESERVED FOR
THE DELNET MONITOR.
4. MAX$NETWORK$CODE WILL INDICATE HOW MANY UNIDS ARE
CURRENTLY OPERATIONAL WITHIN A PARTICULAR COUNTRY.
5. FOR DETAILED INFORMATION ON THE ABOVE REFER TO
PHISTER'S THESIS, APPENDIX D. */

THIS$UNID$NBR LITERALLY '02', /* UNIQUE ADDRESS FOR THIS UNID */
THIS$COUNTRY$CODE LITERALLY '09', /* CC WHERE THIS UNID RESIDES */
MAX$COUNTRY$CODE LITERALLY '09', /* INDICATES COUNTRY CODES IN USE */
MAX$NETWORK$CODE LITERALLY '03', /* INDICATES UNIDS OPERATIONAL IN NET */

/* NOTE: THE LITERAL THIS$COUNTRY$CODE OF 9, ABOVE, IS HERE FOR USE WITH THE

```

```

SOURCE CODE FOR OP544.SRC.

SUMMER QUARTER EE 6.90 PROJECT;I.E., INTERFACE THE UNID II WITH THE NETOS */

DECLARE    L$R$DEST$ERR    LITERALLY '5',    /* LOCAL ROUTE$IN ERROR */
           L$R$DEST$ERR    LITERALLY '6',    /* LOCAL ROUTE$OUT ERROR */

/* MISCELLANEOUS DECLARATIONS */

DECLARE    BUSY            LITERALLY 'OFFH',
           TRUE            LITERALLY 'OFFH',
           FALSE           LITERALLY 'ODH',
           READY           LITERALLY '1',
           DONE            LITERALLY '2',
           CR              LITERALLY 'ODH',
           LF              LITERALLY 'OAH',
           USARTON         LITERALLY '0010$0111B',
           USARTOFF        LITERALLY '0000$0110B',

/*****
/* DATA TABLES USED IN THIS PROGRAM
/*
/*****

DECLARE    SYS$MEM$BASE    LITERALLY '08001H'; /* SYSTEM MEMORY BASE ADDRESS */

DECLARE    SYS$BASE        BYTE AT (SYS$MEM$BASE);

/* DECLARATIONS FOR FLAGS AND TABLES IN SYSTEM MEMORY */

DECLARE    (LSEM$1, LSEM$2, LSEM$3,          /* LOCAL TO NET SEMAPHORE AND */
           LSEM$4, NSEM$1, NSEM$2) BYTE     /* NET TO LOCAL SEMAPHORE */
           AT (.SYS$BASE),

           (LPTR$1, LSPARE$1, LPTR$2, LSPARE$2, /* LOC TO NET PACKET PTR */
           LPTR$3, LSPARE$3, LPTR$4, LSPARE$4,   /* LOC TO NET PACKET PTR */
           NPTR$1, NSPARE$1, NPTR$2, NSPARE$2) ADDRESS /* NET TO LOC PACKET PTR */
           AT (.SYS$BASE + 6),

           LC01RX (PACKET$TABLE$SIZE) BYTE     /* LOCAL BOARD RECEIVE TABLES */
           AT (.SYS$BASE + 30),
           LC02RX (PACKET$TABLE$SIZE) BYTE
           AT (.SYS$BASE + 30 + PACKET$TABLE$SIZE),
           LC03RX (PACKET$TABLE$SIZE) BYTE
           AT (.SYS$BASE + 30 + PACKET$TABLE$SIZE +
               PACKET$TABLE$SIZE),
           LC04RX (PACKET$TABLE$SIZE) BYTE
           AT (.SYS$BASE + 30 + PACKET$TABLE$SIZE +
               PACKET$TABLE$SIZE +
               PACKET$TABLE$SIZE),

           NT01RX (FRAME$TABLE$SIZE) BYTE     /* NETWORK BOARD RECEIVE TABLES */

```

SOURCE CODE FOR OP544.SRC. 25 SEP 85

```

      AT (.SYS$BASE + 30 + PACKET$TABLE$SIZE +
        PACKET$TABLE$SIZE +
        PACKET$TABLE$SIZE +
        PACKET$TABLE$SIZE),
      NT02RX (FRAME$TABLE$SIZE) BYTE
      AT (.SYS$BASE + 30 + PACKET$TABLE$SIZE +
        PACKET$TABLE$SIZE +
        PACKET$TABLE$SIZE +
        PACKET$TABLE$SIZE +
        FRAME$TABLE$SIZE);

```

DECLARE

```

      LC01TX (DATA$TABLE$SIZE) BYTE,
      LC02TX (DATA$TABLE$SIZE) BYTE,
      LC03TX (DATA$TABLE$SIZE) BYTE,
      LC04TX (DATA$TABLE$SIZE) BYTE,

      (LC01NS, LC01NE, LC01SZ,
       LC02NS, LC02NE, LC02SZ,
       LC03NS, LC03NE, LC03SZ,
       LC04NS, LC04NE, LC04SZ) ADDRESS,

      (NT01NS, NT01NE, NT01SZ,
       NT02NS, NT02NE, NT02SZ) ADDRESS,

      (TX01NS, TX01NE, TX01SZ,
       TX02NS, TX02NE, TX02SZ,
       TX03NS, TX03NE, TX03SZ,
       TX04NS, TX04NE, TX04SZ) ADDRESS,

      STAT$B (STAT$NBR) BYTE,

```

```

/*****
/* ADDITIONAL DECLARES NEEDED FOR THIS PROGRAM
*****/

```

/* INTERNAL VARIABLES USED IN THIS MODULE */

```

      (SEND$1, SEND$2, SEND$3, SEND$4) BYTE,

```

```

      (BYTES$SENT$1, BYTES$SENT$2,
       BYTES$SENT$3, BYTES$SENT$4) BYTE,

```

```

      (BYTES$RECV$1, BYTES$RECV$2,
       BYTES$RECV$3, BYTES$RECV$4) BYTE,

```

```

      (RTA$1, TXTR$1, TXTA$1, RXTR$1, RXTA$1,
       RTA$2, TXTR$2, TXTA$2, RXTR$2, RXTA$2,
       RTA$3, TXTR$3, TXTA$3, RXTR$3, RXTA$3,
       RTA$4, TXTR$4, TXTA$4, RXTR$4, RXTA$4) BYTE,

```


25 SEP 85

SOURCE CODE FOR OP544.SRC.

```

DECLARE      TP$6(*) BYTE DATA(CR,LF,
      Completed initializing variables');
DECLARE      TP$7(*) BYTE DATA(CR,LF,
      Completed initializing tables');
DECLARE      TP$8(*) BYTE DATA(CR,LF,
      Entering forever loop!');

DECLARE      MSG1(*) BYTE DATA(CR,LF,
      'Do you want TR/TA handshake (Y/N) [N] ?');
DECLARE      MSG2(*) BYTE DATA(CR,LF,
      'Which channel (1, 2, 3, 4) [1] ?');

/* THE FOLLOWING DECLARE STATEMENTS SET UP THE TRAP, RST5.5, RST6.5,
RST7.5 INTERRUPT TABLE IN EPROM TO JUMP TO LOCATIONS IN THE 8155 RAM,
WHERE ANOTHER JUMP TABLE IS ACCESSABLE BY PROGRAMS IN RAM. THE RAM
TABLE IS INITIALIZED TO JMP 0 WHERE THE EPROM ENTRYJUMP IS LOCATED. */

DECLARE      JUMP$CODE      LITERALLY '0C3H';
DECLARE      VECTOR$TABLE$BASE LITERALLY '7FF4H';

DECLARE      VECTOR$TABLE$BASE$PTR ADDRESS,
      ADDR ADDRESS AT (.VECTOR$TABLE$BASE$PTR),
      VECTOR$TABLE BASED VECTOR$TABLE$BASE$PTR BYTE;

DECLARE      CODE0(3) BYTE AT (0024H) DATA(0C3H,0F4H,7FH),
      CODE1(3) BYTE AT (002CH) DATA(0C3H,0F7H,7FH),
      CODE2(3) BYTE AT (0034H) DATA(0C3H,0FAH,7FH),
      CODE2A(3) BYTE AT (0038H) DATA(0C3H,0.D),
      CODE3(3) BYTE AT (003CH) DATA(0C3H,0FDH,7FH);

/*
DECLARE      INT$VECTOR(8)      ADDRESS,      * INTERRUPT VECTORS *
      TRAP$INT      ADDRESS,      * POWER FAIL SENSE *
      INT$75      ADDRESS,      * TIMER INPUTS, TINT0 & TINT1 *
      INT$65      ADDRESS,      * RING INDICATOR AND CARRIER DETECT *
      INT$55      ADDRESS,      * FLAG (FINT) AND MULTIBUS INTERRUPT *
      RIM$MASK      BYTE,      * INTERRUPT MASK RETURNED BY RIM *
      TRAP$INT$PTR      LITERALLY '024H',      * INTERRUPT LOCATIONS *
      INT$55$PTR      LITERALLY '02CH';
      INT$65$PTR      LITERALLY '034H';
      INT$75$PTR      LITERALLY '03CH';
      */

/*****
*      ISBC 544 I/O DECLARATIONS SECTION
*****
DECLARE      SIM$MASK      LITERALLY '01FH',      /* MASTER/SLAVE PORTS */
      /* SIM MASK-TURN OFF INTERRUPTS */

```

25 SEP 85

SOURCE CODE FOR OP544.SRC.

```
MASTER      /* MASTER MODE */
SLAVE       /* SLAVE MODE */
```

```
DECLARE
  US$P0$CMD  /* 8251 USARTS */
  US$P0$STAT /* SERIAL PORT 0 COMMAND */
  US$P0$DATA /* SERIAL PORT 0 STATUS */
  US$P1$CMD  /* SERIAL PORT 1 COMMAND */
  US$P1$STAT /* SERIAL PORT 1 STATUS */
  US$P1$DATA /* SERIAL PORT 1 DATA */
  US$P2$CMD  /* SERIAL PORT 2 COMMAND */
  US$P2$STAT /* SERIAL PORT 2 STATUS */
  US$P2$DATA /* SERIAL PORT 2 DATA */
  US$P3$CMD  /* SERIAL PORT 3 COMMAND */
  US$P3$STAT /* SERIAL PORT 3 STATUS */
  US$P3$DATA /* SERIAL PORT 3 DATA */
  US$MODE    /* SERIAL PORT MODE */
  US$COMMAND /* SERIAL PORT COMMAND */
```

```
US$RESET$CMD /* RESET USART */
US$DTR$ON    /* RTS,RXE,DTR,TXE */
US$CRT$CMD   /* RTS,ER,RXE,DTR,TXE */
US$TTY$CMD   /* RTS,ER,RXE,TXE */
US$DTR$OFF   /* RTS,RXE,TXE */
US$RXRDY    /* RECIEVER READY */
US$TXE      /* TRANSMITTER EMPTY */
US$TXRDY    /* TRANSMITTER READY */
PARITY$MASK /* MASK OFF PARITY BIT */
```

```
DECLARE
  IT1$CONT /* 8253 INTERVAL TIMER */
  IT1$CNTR0 /* INTERVAL TIMER 1 CONTROL */
  IT1$CNTR1 /* COUNTER 0, USART 0 */
  IT1$CNTR2 /* COUNTER 1, USART 1 */
  IT2$CONT /* COUNTER 2, USART 2 */
  IT2$CNTR0 /* INTERVAL TIMER 2 CONTROL */
  IT2$CNTR1 /* COUNTER 3, USART 3 */
  IT2$CNTR2 /* COUNTER 4, CNTR5 OR SPLIT CLOCKS */
  USART$CNTR$M3 /* COUNTER 5, RST 7.5 */
  /* DIVIDE BY N RATE GENERATOR, MODE 3 */
  /* FOR USART CLK x 16, CLK = 1.2288 MHZ */
```

```
IT1$CNTR0 /* COUNTER 0, USART 0 */
IT1$CNTR1 /* COUNTER 1, USART 1 */
IT1$CNTR2 /* COUNTER 2, USART 2 */
IT2$CNTR0 /* COUNTER 3, USART 3 */
IT2$CNTR1 /* COUNTER 4, CNTR5 OR SPLIT CLOCKS */
IT2$CNTR2 /* COUNTER 5, RST 7.5 */
USART$CNTR$M3 /* COUNTER 5, RST 7.5 */
/* DIVIDE BY N RATE GENERATOR, MODE 3 */
/* FOR USART CLK x 16, CLK = 1.2288 MHZ */
```

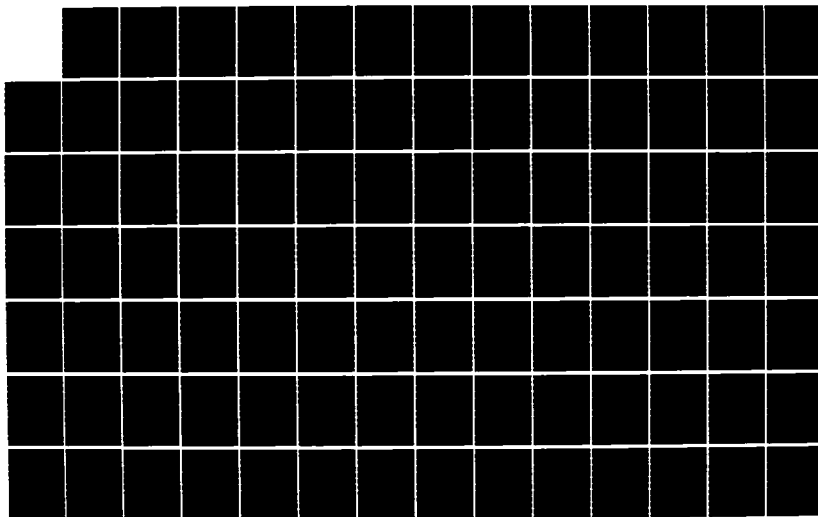
AD-A164 842

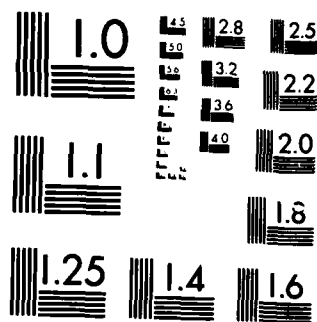
DEVELOPMENT AND IMPLEMENTATION OF THE X25 PROTOCOL FOR
THE UNIVERSAL METH. (U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI. . M W WEBER
DEC 85 AFIT/GE/ENG/85D-52-VOL-2 F/G 17/2

2/3

UNCLASSIFIED

NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

25 SEP 85

SOURCE CODE FOR OP544.SRC.

```

      B110      LITERALLY '698'; /* TIMER VALUE FOR 110 BPS */

DECLARE
  PI$PORTA      LITERALLY '0E9H'; /*8155 PERIPHERAL INTERFACE*/
  PI$PORTB      LITERALLY '0EAH'; /* PORT A (OUTPUT) */
  PI$PORTC      LITERALLY '0EBH'; /* PORT B (INPUT) */
  PI$STAT      LITERALLY '0EBH'; /* PORT C (INPUT) */
  PI$CMD      LITERALLY '0EBH'; /* PPI STATUS */
  PI$CNTR$LO      LITERALLY '0EDH'; /* PPI COMMAND */
  PI$CNTR$HI      LITERALLY '0EDH'; /* PPI COUNTER LO BYTE */
  PI$CNTR$LOCNT      LITERALLY '16384'; /* PPI COUNTER HI BYTE */
  PI$CNTR$HICNT      LITERALLY '75'; /* PPI COUNTER TIME CONST */
  PI$INIT$CMD1      LITERALLY '041H'; /* PPI INITIALIZATION COMMAND 1 */
  PI$INIT$CMD2      LITERALLY '0C1H'; /* A OUT, B & C IN, STOP COUNT */
  PI$INIT$US$INT1      LITERALLY '0F0H'; /* PPI INITIALIZATION COMMAND 2 */
  PI$INIT$US$INT2      LITERALLY '0C0H'; /* A OUT, B & C IN, START COUNT */
  PI$PORTC$STAT      LITERALLY '0CCH'; /* PORT C STATUS */
  PI$PORTC$CTL      LITERALLY '0CEH'; /* PORT C CONTROL */
  PI$M2M1      LITERALLY '0C6H'; /* A-MODE 1, B-MODE 2 */
  PI$0BF      LITERALLY '0B0H'; /* OUTPUT BUFFER READY */
  PI$1BF      LITERALLY '02H'; /* INPUT BUFFER READY */

/*8259 INTERRUPT CONTROLLER*/
DECLARE
  IC$PORTA      LITERALLY '0E6H'; /* PORT A */
  IC$PORTB      LITERALLY '0E7H'; /* PORT B */
  IC$ICW1      LITERALLY '56H'; /* INIT COMMAND WORD 1,
  (A7A6A5) = 010; EDGE TRIG;
  INTERVAL = 4; SINGLE; NO ICW4 */
  IC$ICW2      LITERALLY '00H'; /* INIT COMMAND WORD 2,
  (A15-A8) = 0 */
  IC$ICW3      LITERALLY '00H'; /* INIT COMMAND WORD 3,
  NO SLAVE IN IR */
  IC$ICW4      LITERALLY '0CH'; /* INIT COMMAND WORD 4, NOT USED */
  IC$MASK      LITERALLY '0'; /* INTERRUPT MASK, OCW1, NOT USED;
  0 = ALL INTERRUPTS ENABLED */
  INIT$MASK      LITERALLY '10101010B'; /* INITIAL INTERRUPT MASK, OCW1;
  RECEIVE INTR ON, TRANSMIT INTR OFF */
  IC$EOI      LITERALLY '0A0H'; /* END OF INTERRUPT CMD, OCW2.
  ROTATE (PRIORITY) ON NON-SPECIFIC EOI */
  IC$OCW3IR      LITERALLY '0AH'; /* READ IRR LEVEL, OCW3.
  NO SPECIAL MASK MODE; NO POLL;
  READ IR REG (NOT USED) */
  IC$OCW3IS      LITERALLY '0BH'; /* READ ISR LEVEL, OCW3.
  NO SPECIAL MASK MODE; NO POLL;

```

25 SEP 85

SOURCE CODE FOR OP544.SRC.

```

        READ IS REG (NOT USED) */
        /* SPECIAL MASK MODE SET */
        /* SPECIAL MASK MODE RESET */
        IC$OCW3$SMMS LITERALLY '60H';
        IC$OCW3$SMR LITERALLY '40H';

/*****
* DATE: 24 SEP 85
* VERSION: 1.2
* NAME: INIT$NSTAB
* MODULE NUMBER: 2.0
* DESCRIPTION: Initializes the hardware devices found on the
                SBC 544 (8255, 8255As, 8259, 8253s ). All ports may
                be used for communications with other hosts.
* PASSED VARIABLES: None
* RETURNS: None
* GLOBAL VARIABLES USED: BRF0, BRF1, BRF2, BRF3, ISR$STAT
* GLOBAL VARIABLES CHANGED: All the above
* MODULES CALLED: None
* CALLING MODULES: main
* AUTHOR: Capt C.T. Childress
* HISTORY:
*          1.0 Capt C.T. Childress - original version
*****/

        /* INITIALIZE 8155 PROGRAMMABLE PERIPHERAL INTERFACE (PPI) */
        /* SET DATA RATES FOR LOCAL CHANNEL USARTS: */
        /* SET PIT #1, REGISTER #0 FOR USART #0 DATA RATE */
        /* SET PIT #1, REGISTER #1 FOR USART #1 DATA RATE */
        /* SET PIT #1, REGISTER #2 FOR USART #2 DATA RATE */
        /* SET PIT #2, REGISTER #0 FOR USART #3 DATA RATE */
        /* INITIATE RESET ON ALL USARTS */
        /* INITIALIZE THE FOUR USARTS TO: 2 STOP BITS, NO PARITY, */
        /* 8 BIT CHARACTERS, AND 16X DATA RATE */
        /* INITIALIZE THE 8259 PROGRAMMABLE INTERRUPT CONTROLLER */

/*****
* DEVICE INITIALIZATION SECTION
*****/

INITIALIZ$BOARD:
/* THIS PROCEDURE INITIALIZES THE 158C 544 BOARD */
PROCEDURE:

P$INIT: /* 8155 PARALLEL INTERFACE INITIALIZATION */
        OUTPUT(P$CMD0) = P$INIT$CMD0;
        OUTPUT(P$PORTA) = P$INIT$USINT1; /* RESETS THE 8251 USARTS */
        OUTPUT(P$PORTA) = P$INIT$USINT2;
        OUTPUT(P$CNTR$LO) = P$CNTR$LOCNT;
        OUTPUT(P$CNTR$HI) = P$CNTR$HICNT;

```

25 SEP 85

SOURCE CODE FOR OP544.SRC.

OUTPUT(PICMD) = P\$INIT\$CMD2;

IT\$INIT: /* 8253 INTERVAL TIMER INITIALIZATION */

```

    BRFO = B9600;
    BRF1 = B9600;
    BRF2 = B9600;
    BRF3 = B9600;
    OUTPUT(IT1$CONT) = USART$CNTR$M3; /* USART 0 */
    OUTPUT(IT1$CENTRO) = LOW(BRF0);
    OUTPUT(IT1$CENTRO) = HIGH(BRF0);
    OUTPUT(IT1$CONT) = 40H OR USART$CNTR$M3; /* USART 1 */
    OUTPUT(IT1$CENTR1) = LOW(BRF1);
    OUTPUT(IT1$CENTR1) = HIGH(BRF1);
    OUTPUT(IT1$CENTR1) = 80H OR USART$CNTR$M3; /* USART 2 */
    OUTPUT(IT1$CENTR2) = LOW(BRF2);
    OUTPUT(IT1$CENTR2) = HIGH(BRF2);
    OUTPUT(IT1$CENTR2) = USART$CNTR$M3; /* USART 3 */
    OUTPUT(IT2$CENTRO) = LOW(BRF3);
    OUTPUT(IT2$CENTRO) = HIGH(BRF3);

```

IC\$INIT: /* 8259 INTERRUPT CONTROLLER INITIALIZATION */

/*THE FOLLOWING CODE INITIALIZES THE 8259A. THE INTERRUPTS ARE USED FOR THIS PROGRAM. ALL INTERRUPTS ARE ON EXCEPT FOR THE RST TYPES */

```

    OUTPUT(IC$PORTA) = IC$ICW1;
    OUTPUT(IC$PORTB) = IC$ICW2;
    OUTPUT(IC$PORTB) = INIT$MASK; /* OCW1 */
    OUTPUT(IC$PORTA) = IC$EOI; /* OCW2 */
    OUTPUT(IC$PORTA) = IC$OCW3IS; /* OCW3 */
    ISR$STAT = INPUT(IC$PORTA);

```

US\$INIT: /* INITIALIZE THE USARTS */
 /* ASYNC MODE, 8 DATA BITS, 1 STOP BIT, NO PARITY, x16 CLOCK */
 DTR, RTS ON, RECV ON, TRANSMIT OFF

```

    OUTPUT(US$P0$CMD) = US$MODE;
    OUTPUT(US$P0$CMD) = USARTON;
    OUTPUT(US$P1$CMD) = US$MODE;
    OUTPUT(US$P1$CMD) = USARTON;
    OUTPUT(US$P2$CMD) = US$MODE;
    OUTPUT(US$P2$CMD) = USARTON;
    OUTPUT(US$P3$CMD) = US$MODE;
    OUTPUT(US$P3$CMD) = USARTON;

```

END INITIALIZES\$BOARD;

```

/*****
* DATE: 24 SEP 85
* VERSION: 1.1
* NAME: INIT
*****/

```

25 SEP 85

SOURCE CODE FOR OP544.SRC.

```

* MODULE NUMBER: 4.0
* DESCRIPTION: Initializes the pointers and tables used with
* the SBC 544 software. The vector table may be used
* when UNID II is ready for full operation. Presetly,
* the interrupt table is located in the 256 bytes of RAM
* available to the 8255. This allows modification of the
* table without changing EPROMS. For this implementation
* the interrupt table is commented out.
* PASSED VARIABLES: None
* RETURNS: None
* GLOBAL VARIABLES USED: VECTOR TABLE, BYTESSENT$1, BYTESSENT$2,
* BYTESSENT$3, BYTESSENT$4,
* BYTES$RCV$1, BYTES$RCV$2, BYTES$RCV$3,
* BYTES$RCV$4, SEND$F, SEND$2, SEND$3, SEND$4,
* TRTA$1, TRTA$2, TRTA$3, TRTA$4, TXTR$1, TXTR$2,
* TXTR$3, TXTR$4, TXTA$1, TXTA$2, TXTA$3, TXTA$4,
* CHAR$1, CHAR$2, CHAR$3, CHAR$4, CHAR$X
* GLOBAL VARIABLES CHANGED: All the above
* MODULES CALLED: None
* CALLING MODULES: main
* AUTHOR: Capt C.T. Childress
* HISTORY: 1.1 Capt C.T. Childress - original 544 version
*****
INIT: PROCEDURE;
DECLARE I BYTE;

/* THE FOLLOWING CODE INITIALIZES THE JUMP TABLE IN 8155 RAM TO POINT
TO THE USER'S INTERRUPT SERVICE ROUTINES AT INTR$VECTOR$5$5, ETC. */

/* ADDR = .INTR$VECTOR$5$5;
VECTOR$TABLE = JUMP$CODE;
ADDR = ADDR + 1;
VECTOR$TABLE = LOW(VECTOR$TABLE$BASE);
ADDR = ADDR + 1;
VECTOR$TABLE = HIGH(VECTOR$TABLE$BASE);

ADDR = .INTR$VECTOR$6$5;
VECTOR$TABLE = JUMP$CODE;
ADDR = ADDR + 1;
VECTOR$TABLE = LOW(VECTOR$TABLE$BASE + 3);
ADDR = ADDR + 1;
VECTOR$TABLE = HIGH(VECTOR$TABLE$BASE + 3);

ADDR = .INTR$VECTOR$7$5;
VECTOR$TABLE = JUMP$CODE;
ADDR = ADDR + 1;
VECTOR$TABLE = LOW(VECTOR$TABLE$BASE + 6);
ADDR = ADDR + 1;

```

SOURCE CODE FOR OP544.SRC.

25 SEP 85

PAGE 13

VECTOR\$TABLE = HIGH(VECTOR\$TABLE\$BASE + 6);

/* INITIALIZES 8155 JUMP TABLE TO JMP 0 */

```
I = 0;
ADDR = VECTOR$TABLE$BASE;      /* = 7FF4H */
DO WHILE I <= 11;
  VECTOR$TABLE = JUMP$CODE;
  ADDR = ADDR + 1;
  VECTOR$TABLE = 0;
  ADDR = ADDR + 1;
  VECTOR$TABLE = 0;
  ADDR = ADDR + 1;
  I = I + 3;
END;
```

```
BYTE$SENT$1 = 0;
BYTE$SENT$2 = 0;
BYTE$SENT$3 = 0;
BYTE$SENT$4 = 0;
```

```
BYTE$RECV$1 = 0;
BYTE$RECV$2 = 0;
BYTE$RECV$3 = 0;
BYTE$RECV$4 = 0;
```

```
SEND$1 = FALSE;
SEND$2 = FALSE;
SEND$3 = FALSE;
SEND$4 = FALSE;
```

```
TRTA$1, TXTR$1, TXTA$1, RXTR$1, RXTA$1 = FALSE;
TRTA$2, TXTR$2, TXTA$2, RXTR$2, RXTA$2 = FALSE;
TRTA$3, TXTR$3, TXTA$3, RXTR$3, RXTA$3 = FALSE;
TRTA$4, TXTR$4, TXTA$4, RXTR$4, RXTA$4 = FALSE;
```

```
CHAR$1 = '1';
CHAR$2 = '2';
CHAR$3 = '3';
CHAR$4 = '4';
CHAR$X = 'X';
```

END INIT;

```
/* *****
* DATE: 2 OCT 84
* VERSION: 1.1
* NAME: INIT$TAB
* *****
```

25 SEP 85

SOURCE CODE FOR OP544.SRC.

```

* MODULE NUMBER: 5.0
* DESCRIPTION: Initializes the pointers and tables used with
* the SBC 544 software.
* PASSED VARIABLES: None
* RETURNS: None
* GLOBAL VARIABLES USED: LSEM$1, LSEM$2, LSEM$3, LSEM$4, NSEM$1,
* NSEM$2, LSPARE$1, LSPARE$2, LSPARE$3, LSPARE$4,
* LPTR$1, LPTR$2, LPTR$3,
* LPTR$4, LC01NS, LC02NS, LC03NS, LC04NS, LC01NS, LC02NS,
* LC03NS, LC04NS, LC01NS, LC02NS, LC03NS, LC04NS,
* LC02NE, NT01NS, NT02NS, NT03NS, NT04NS, NT01NS, NT02NS,
* NT03NS, NT04NS, TX01NS, TX02NS, TX03NS, TX04NS, TX01NS,
* TX02NS, TX03NS, TX04NS, TX01NS, TX02NS, TX03NS, TX04NS,
* TX02NE, TX03NE, TX04NE, TX01NS, TX02NS, TX03NS, TX04NS,
* TX04SZ, STATTB, STAT$NBR
* GLOBAL VARIABLES CHANGED: All the above
* MODULES CALLED: None
* CALLING MODULES: main
* AUTHOR: Capt C.T. Childress
* HISTORY:
* 1.1 Capt C.T. Childress - original 544 version
*****
INIT$TAB: PROCEDURE:
DECLARE IX BYTE;

LSEM$1 = DONE; /* INITIALIZATION OF THE SEMAPHORE FLAGS */
LSEM$2 = DONE;
LSEM$3 = DONE;
LSEM$4 = DONE;

NSEM$1 = DONE;
NSEM$2 = DONE;

LSPARE$1 = 0; /* THE SPARE ADDRESS LOCATIONS SHOULD BE SET TO THE
LSPARE$2 = 0; /* SEGMENT NUMBER INTO WHICH THE LOCAL BOARD MEMORY
LSPARE$3 = 0; /* WILL BE MAPPED INTO SYSTEM MEMORY. THIS VALUE IS
LSPARE$4 = 0; /* CURRENTLY 0 FOR THE LSPARE$X VARIABLES.

NSPARE$1 = 0; /* SAME AS FOR LSPARE$X DISCUSSION ABOVE
NSPARE$2 = 0;

LPTR$1 = 0;
LPTR$2 = 0;
LPTR$3 = 0;
LPTR$4 = 0;

NPTR$1 = 0;
NPTR$2 = 0;

NT01NS = 0;

```

25 SEP 85

SOURCE CODE FOR OPS44.SRC.

```

* VERSION: 1.1
* NAME: SNDSEQ
* MODULE NUMBER: 3.0
* DESCRIPTION: Sends messages to port 0 of the SBC 544 board
* for a local monitor. The procedure checks the output
* buffer status in a loop until the buffer is empty, then
* places one byte at a time until the message is
* finished.
* PASSED VARIABLES: MSG, TOTAL
* RETURNS: None
* GLOBAL VARIABLES USED: None
* GLOBAL VARIABLES CHANGED: None
* MODULES CALLED: None
* CALLING MODULES: main
* AUTHOR: Capt C.T. Childress
* HISTORY:
* 1.1 Capt C.T. Childress - original 544 version
*
*****
SNDSEQ: PROCEDURE(MSG, TOTAL);
  DECLARE MSG ADDRESS;
  CHAROUT BASED MSG(1) BYTE;
  DECLARE (COUNT, TOTAL) BYTE;

  COUNT = 0;
  LOOP: DO WHILE COUNT < TOTAL;
    DO WHILE NOT (INPUT(US$PO$STAT) AND 1); END;
    OUTPUT(US$PO$DATA) = CHAROUT(COUNT);
    COUNT = COUNT + 1;
  END LOOP;

```

```

END SNDSEQ;
/*****
* DATE: 2 OCT 84
* VERSION: 1.1
* NAME: LD$TAB$H$SKP
* MODULE NUMBER: 7.2.2
* DESCRIPTION: This procedure houskeeps the specified buffer
* after loading th user data from the host. This action
* requires updating the NE pointer of the specified table.
* When a pointer comes to the end of a table it is looped
* back to the beginning.
* PASSED VARIABLES: TABLE
* RETURNS: None
* GLOBAL VARIABLES USED: LC01NE
* LC02NE, LC03NE, LC04NE, TX01NE, TX02N3, TX03N3,
* TX04NE, NT01NE, NT02NE
* GLOBAL VARIABLES CHANGED: All the above
* MODULES CALLED: None
* CALLING MODULES: main

```

25 SEP 85

SOURCE CODE FOR OP544.SRC.

```
NT01NE = 0;
NT01SZ = FRAME$TABLE$SIZE;

NT02NS = 0;
NT02NE = 0;
NT02SZ = FRAME$TABLE$SIZE;

LC01NS = 0;
LC01NE = 0;
LC01SZ = PACKET$TABLE$SIZE;

LC02NS = 0;
LC02NE = 0;
LC02SZ = PACKET$TABLE$SIZE;

LC03NS = 0;
LC03NE = 0;
LC03SZ = PACKET$TABLE$SIZE;

LC04NS = 0;
LC04NE = 0;
LC04SZ = PACKET$TABLE$SIZE;

TX01NS = 0;
TX01NE = 0;
TX01SZ = DATA$TABLE$SIZE;

TX02NS = 0;
TX02NE = 0;
TX02SZ = DATA$TABLE$SIZE;

TX03NS = 0;
TX03NE = 0;
TX03SZ = DATA$TABLE$SIZE;

TX04NS = 0;
TX04NE = 0;
TX04SZ = DATA$TABLE$SIZE;

IX = 0;
DO WHILE IX < STAT$NBR;
  STATB(IX) = 0;
  IX = IX + 1;
END;
```

END INIT\$TAB;

```
/*.....
* DATE: 2 OCT 84
*.....
```


SOURCE CODE FOR OP544.SRC.

25 SEP 85

```

* AUTHOR:
*
* HISTORY:
*      1.1 Capt C.T. Childress - original 544 version
*      *****
LD$TAB$SKP: PROCEDURE(TABLE);
DECLARE TABLE BYTE;

```

Capt C.T. Childress

```

IF (TABLE >= 1 AND TABLE <= 10) THEN /* 8 FOR REAL, 10 FOR SIM */
DO CASE TABLE;

```

```

; /* CASE ZERO IS NULL */

```

```

/* CASES 1, 2, 3, 4 FOR LC01(2,3,4) ARE NOT REQUIRED AS */
/* THE LCOXNS POINTERS ARE INCREMENTED AND CHECKED IN THE */
/* RECEIVE DATAGRAM INTERRUPT ROUTINES. USE FOR SIMULATION */
/* PURPOSES ONLY. CASE 9 IS FOR SIMULATION ONLY */
*/

```

```

DO;
  LC01NE = LC01NE + PACKET$SIZE;
  IF LC01NE >= LC01SZ THEN
    LC01NE = 0;

```

END;

```

DO;
  LC02NE = LC02NE + PACKET$SIZE;
  IF LC02NE >= LC02SZ THEN
    LC02NE = 0;

```

END;

```

DO;
  LC03NE = LC03NE + PACKET$SIZE;
  IF LC03NE >= LC03SZ THEN
    LC03NE = 0;

```

END;

```

DO;
  LC04NE = LC04NE + PACKET$SIZE;
  IF LC04NE >= LC04SZ THEN
    LC04NE = 0;

```

END;

```

DO;
  TX01NE = TX01NE + DATA$GRAM$SIZE;
  IF TX01NE >= TX01SZ THEN
    TX01NE = 0;

```

END;

```

DO;
  TX02NE = TX02NE + DATA$GRAM$SIZE;

```

25 SEP 85

SOURCE CODE FOR OP544.SRC.

```

      IF TX02NE >= TX02SZ THEN
        TX02NE = 0;
      END;

      DO;
        TX03NE = TX03NE + DATA$GRAM$SIZE;
        IF TX03NE >= TX03SZ THEN
          TX03NE = 0;
        END;

      DO;
        TX04NE = TX04NE + DATA$GRAM$SIZE;
        IF TX04NE >= TX04SZ THEN
          TX04NE = 0;
        END;

      DO;
        NT01NE = NT01NE + FRAME$SIZE;
        IF NT01NE >= NT01SZ THEN
          NT01NE = 0;
        END;

      DO;
        NT02NE = NT02NE + FRAME$SIZE;
        IF NT02NE >= NT02SZ THEN
          NT02NE = 0;
        END;

      END;
    ELSE
      DO;
        STATTB(7) = STATTB(7) + 1;
        STATTB(0) = STATTB(0) + 1;
      END;

    END LD$TAB$H$SKP;
  
```

```

/*****
* DATE: 2 OCT 84
* VERSION: 1.1
* NAME: SRVC$TAB$H$SKP
* MODULE NUMBER: 7.4
* DESCRIPTION: This procedure houskeeps the specified buffer
* after serving the table. This action updates the
* next to service (NS) pointer of the specified table.
* When a pointer comes to the end of a table it is looped
* back to the beginning.
* PASSED VARIABLES: TABLE
*

```

```

* RETURNS:
* GLOBAL VARIABLES USED:      None
*                             LC01NS
*                             LC02NS, LC03NS, LC04NS, TX01NS, TX02N3, TX03N3,
*                             TX04NS, NT01NS, NT02NS
* GLOBAL VARIABLES CHANGED:  All the above
* MODULES CALLED:           None
* CALLING MODULES:          main
* AUTHOR:                   Capt C.T. Childress
* HISTORY:
*                             1.1 Capt C.T. Childress - original 544 version
*                             *****
* SRVCTAB$H$SKP: PROCEDURE(TABLE) ;
* DECLARE TABLE BYTE;

```

```

IF (TABLE >= 1 AND TABLE <= 10) THEN /* 8 FOR REAL, 10 FOR SIM */
DO CASE TABLE;

```

```

; /* CASE ZERO IS NULL */

```

```

DO;
  LC01NS = LC01NS + PACKET$SIZE;
  IF LC01NS >= LC01SZ THEN
    LC01NS = 0;
END;

```

```

DO;
  LC02NS = LC02NS + PACKET$SIZE;
  IF LC02NS >= LC02SZ THEN
    LC02NS = 0;
END;

```

```

DO;
  LC03NS = LC03NS + PACKET$SIZE;
  IF LC03NS >= LC03SZ THEN
    LC03NS = 0;
END;

```

```

DO;
  LC04NS = LC04NS + PACKET$SIZE;
  IF LC04NS >= LC04SZ THEN
    LC04NS = 0;
END;

```

```

/* CASES 5, 6, 7, 8 FOR TX01(2,3,4) ARE NOT REQUIRED AS */
/* THE TX0XNS POINTERS ARE INCREMENTED AND CHECKED IN THE */
/* TRANSMIT DATAGRAM INTERRUPT ROUTINES. USE FOR SIMULATION */
/* PURPOSES ONLY. CASE 9 IS FOR SIMULATION ONLY */
*/

```

```

DO;

```

```
TX01NS = TX01NS + DATA$GRAM$SIZE;  
IF TX01NS >= TX01SZ THEN  
  TX01NS = 0;  
END;
```

```
DO;  
  TX02NS = TX02NS + DATA$GRAM$SIZE;  
  IF TX02NS >= TX02SZ THEN  
    TX02NS = 0;  
END;
```

```
DO;  
  TX03NS = TX03NS + DATA$GRAM$SIZE;  
  IF TX03NS >= TX03SZ THEN  
    TX03NS = 0;  
END;
```

```
DO;  
  TX04NS = TX04NS + DATA$GRAM$SIZE;  
  IF TX04NS >= TX04SZ THEN  
    TX04NS = 0;  
END;
```

```
DO;  
  NT01NS = NT01NS + FRAME$SIZE;  
  IF NT01NS >= NT01SZ THEN  
    NT01NS = 0;  
END;
```

```
DO;  
  NT02NS = NT02NS + FRAME$SIZE;  
  IF NT02NS >= NT02SZ THEN  
    NT02NS = 0;  
END;
```

```
END;
```

```
ELSE
```

```
  DO;  
    STATTB(8) = STATTB(8) + 1;  
    STATTB(0) = STATTB(0) + 1;  
  END;
```

```
END SRVC$TAB$H$K$P;
```

```
/*  
*****  
/* INCLUDES THE INTERRUPT DRIVER ROUTINES */  
$ INCLUDE(INTR3.LOC)  
*****  
*/
```

25 SEP 85

SOURCE CODE FOR OP544.SRC.

```

*****
* DATE: 2 OCT 84
* VERSION: 1.1
* NAME: SEND$PACKET
* MODULE NUMBER: 7.3
* DESCRIPTION: This procedure receives an address that
* indicates the local input buffer where the incoming
* host data is located, then formats a packet for
* transmission to the datalink layer. This implementation
* changes Phister's format. Ten bytes are used the
* packet header instead of five bytes. See Appendix G
* for the current packet format.
* Byte 0: GFI / LCGN
* Byte 1: LCN
* Byte 2: Packet Sequence Number
* Byte 3: Source Length and Destination Length
* Byte 4: Source Address
* Byte 5: Destination Address
* Byte 6: Padding byte (total padding = 10 bits
* Byte 7: 2 bits padding and Facility Length
* Byte 8: Facility Code
* Byte 9: Facility Parameter
*
* PASSED VARIABLES: TABLE$PTR, PORT
* RETURNS: None
* GLOBAL VARIABLES USED: LPTR$1, LPTR$2, LSEM$1, LSEM$2, LSEM$3,
* LSEM$4, LPTR$3, LPT4$4
* GLOBAL VARIABLES CHANGED: All the above
* MODULES CALLED: None
* CALLING MODULES: main
* AUTHOR: Capt C.T. Childress
*
* HISTORY: 1.1 Capt C.T. Childress - original 544 version
*****
SEND$PACKET: PROCEDURE(TABLE$PTR, PORT);
DECLARE TABLE$PTR ADDRESS,
(PORT) BYTE,
LCOXTB BASED TABLE$PTR (1) BYTE;

LCOXTB( 0) = 010H; /* GFI/LCGN */
LCOXTB( 1) = (SOURCE$ADDRESS AND OF0H) OR (SHL(DESTINATION$ADDRESS, 4) /*
AND OF0H); /* LCN */
LCOXTB( 2) = 0; /* SEQUENCE NUMBER */
LCOXTB( 3) = 044H; /* SRC, DEST FIELD LENGTH */
LCOXTB( 4) = SOURCE$ADDRESS;
LCOXTB( 5) = DESTINATION$ADDRESS;
LCOXTB( 6) = 0; /* PADDING */
LCOXTB( 7) = 010H; /* FACILITY FIELD LENGTH */
LCOXTB( 8) = 0; /* FACILITY CODE */
LCOXTB( 9) = 0; /* FACILITY PARAMETER */

```

25 SEP 85

SOURCE CODE FOR OP544.SRC.

```
DO CASE PORT;
;
```

```
DO;
LPTR$1 = TABLE$PTR;
LSEM$1 = READY;
END;
```

```
DO;
LPTR$2 = TABLE$PTR;
LSEM$2 = READY;
END;
```

```
DO;
LPTR$3 = TABLE$PTR;
LSEM$3 = READY;
END;
```

```
DO;
LPTR$4 = TABLE$PTR;
LSEM$4 = READY;
END;
```

```
END; /* CASE */
```

END SEND\$PACKET;

```

/*****
* DATE: 2 OCT 84
* VERSION: 1.1
* NAME: DET$ADDR
* MODULE NUMBER: 7.1
* DESCRIPTION: This module determines the destination of data from the local host. A pointer is passed to the routine which gives the location of the packet to be evaluated. The routine processes the control byte, the COUNTRY$CODE, and NETWORK$CODE of the packet. The routine returns "LN" for local to network, "LL" for local to local, and the DESTINATION-ADDRESS. An example of a destination address is 21 for UNID = 2, Channel = 1.
* PASSED VARIABLES: TABLE$PTR
* RETURNS: None
* GLOBAL VARIABLES USED: DESTINATION, DESTINATION$ADDRESS,
* STATTB
* GLOBAL VARIABLES CHANGED: All the ones above
* MODULES CALLED: None
* CALLING MODULES: ROUTESIN
* AUTHOR: C.T. Childress
*****/

```

25 SEP 85

SOURCE CODE FOR OP544.SRC.

```

* HISTORY:  1.1 C.T. Childress - 30 SEP 85 - original SRC 544
*
* ***** implementation *****
DET$ADDR: PROCEDURE(TABLE$PTR);
DECLARE LOBITS BYTE,
        HIBITS BYTE,
        IPCNTL BYTE,
        CONTROL$CODE BYTE,
        COUNTRY$CODE BYTE,
        NETWORK$CODE BYTE,
        HOST$CODE BYTE,
        SRC$HOST$CODE BYTE,
        SRC$NET$CODE BYTE,
        SRC$CONTR$CODE BYTE,
        SRC$CONTRV$CODE BYTE,
        TABLE$PTR ADDRESS,
        LCOXR$BASED TABLE$PTR(1) BYTE;

LOBITS = 0;
HIBITS = 0;
IPCNTL = 0;
CONTROL$CODE = 0;
COUNTRY$CODE = 0;
NETWORK$CODE = 0;
HOST$CODE = 0;
DESTINATION$ADDRESS = 0;
SOURCE$ADDRESS = 0;
DESTINATION = -1;

IPCNTL = LCOXR(1) AND 0E0H;
IF IPCNTL = 0 THEN
DO;

CONTROL$CODE = ROR(LCOXR(16), 4) AND 0FH;
IF CONTROL$CODE = 00 THEN
DO;
COUNTRY$CODE = LCOXR(16) AND 0FH;
IF COUNTRY$CODE = THIS$COUNTRY$CODE THEN /* ELSE TO UNID 0 */
DO;
NETWORK$CODE = ROR(LCOXR(17), 4) AND 0FH;
IF (NETWORK$CODE <= MAX$NETWORK$CODE) THEN
DO;
IF (COUNTRY$CODE <> THIS$COUNTRY$CODE) OR
((NETWORK$CODE) <> THIS$UNID$NBR) THEN
DESTINATION = 1; /* LOC TO NET */
ELSE
DESTINATION = 0; /* LOC TO LOC */
LOBITS = (ROR(LCOXR(18), 4) AND 0FH);

```

```

HIBITS = (ROL(LCOXRX(17), 4) AND OFOH);
HOST$CODE = LOBITS OR HIBITS;
IF (HOST$CODE >= 0) AND (HOST$CODE <= 63) THEN
  DESTINATION$ADDRESS = (ROL(NETWORK$CODE, 4) AND OFOH) OR 1;
ELSE
  IF (HOST$CODE >= 64) AND (HOST$CODE <= 127) THEN
    DESTINATION$ADDRESS = (ROL(NETWORK$CODE, 4) AND OFOH) OR 2;
  ELSE
    IF (HOST$CODE >= 128) AND (HOST$CODE <= 191) THEN
      DESTINATION$ADDRESS = (ROL(NETWORK$CODE, 4) AND OFOH) OR 3;
    ELSE
      IF (HOST$CODE >= 192) AND (HOST$CODE <= 255) THEN
        DESTINATION$ADDRESS = (ROL(NETWORK$CODE, 4) AND OFOH) OR 4;
      END;
    END;
  DO; /* NOT WITHIN NETWORK CODES FOR THIS$COUNTRY$CODE */
    STATTB(4) = STATTB(4) + 1;
  END;
END;
DO; /* NOT FOR THIS COUNTRY, SEND TO NETWORK/UNID 0 */
  STATTB(3) = STATTB(3) + 1;
END;
END;
ELSE
  DO; /* IT IS AT THIS POINT THAT OTHER X.121 CONTROL CODES
        SOFTWARE SUPPORT WILL BE INCORPORATED INTO THE NETWORK */
    STATTB(2) = STATTB(2) + 1;
  END;
END;
END;
ELSE
  IF IPCNTL = 0COH THEN
    DO;
      /* IP CONTROL; LOOK AT SUBCODE, CHECK SUBCODE FOR SQUELCH
        OR UNSQUELCH AND ACT ACCORDINGLY */
    END;
  ELSE
    DO;
      DESTINATION = -1;
      STATTB(10) = STATTB(10) + 1;
    END;
  END;
END;
IF DESTINATION = 1 THEN /* LOC TO NET, GET SOURCE INFO */
  DO;
    SRC$NET$CODE = ROR(LCOXRX(13), 4) AND OFH;
    IF (SRC$NET$CODE = THIS$UNID$NBR) THEN
      DO;
        LOBITS = (ROR(LCOXRX(14), 4) AND OFH);

```



```

HIBITS = (ROL(LCOXR(13), 4) AND OFOH);
SRC$HOST$CODE = LOBITS OR HIBITS;
IF (SRC$HOST$CODE >= 0) AND (SRC$HOST$CODE <= 63) THEN
  SOURCE$ADDRESS = (ROL(SRC$NET$CODE, 4) AND OFOH) OR 1;
ELSE
  IF (SRC$HOST$CODE >= 64) AND (SRC$HOST$CODE <= 127) THEN
    SOURCE$ADDRESS = (ROL(SRC$NET$CODE, 4) AND OFOH) OR 2;
  ELSE
    IF (SRC$HOST$CODE >= 128) AND (SRC$HOST$CODE <= 191) THEN
      SOURCE$ADDRESS = (ROL(SRC$NET$CODE, 4) AND OFOH) OR 3;
    ELSE
      IF (SRC$HOST$CODE >= 192) AND (SRC$HOST$CODE <= 255) THEN
        SOURCE$ADDRESS = (ROL(SRC$NET$CODE, 4) AND OFOH) OR 4;
      END;
    END;
  END;
DO;
  DESTINATION = -1;
  STATTB(9) = STATTB(9) + 1;
END;

```

END;

```

IF DESTINATION = -1 THEN
  STATTB(0) = STATTB(0) + 1;

```

END DET\$ADDR;

```

/*****
* DATE: 2 OCT 84
* VERSION: 1.1
* NAME: DET$DEST$NL
* MODULE NUMBER: 7.5
* DESCRIPTION: This module determines the destination from the second byte of the packet header.
* PASSED VARIABLES: TABLE$PTR
* RETURNS: PORT
* GLOBAL VARIABLES USED: NTOXR, STATTB.
* GLOBAL VARIABLES CHANGED: NTOXR, STATTB.
* MODULES CALLED: SENDSEQ
* CALLING MODULES: ROUTE$IN
* AUTHOR: C. T. Childress
* HISTORY: 1.1 C.T. Childress - 2 OCT 84 - original SBC 544
* implementation.
*****
DET$ADDR$NL: PROCEDURE(TABLE$PTR) BYTE;

```

```

  DECLARE PORT BYTE,
    TABLE$PTR ADDRESS,
    NTOXR BASED TABLE$PTR (1) BYTE;

```

SOURCE CODE FOR OP544.SRC, 25 SEP 85

```

PORT = NTOXR(X(0)) AND OFH;
IF (PORT >= 1 AND PORT <= 4) THEN
  RETURN PORT;
ELSE
  DO;
    PORT = -1;
    STATB(1) = STATB(1) + 1; /* INCREMENT LOCAL ERROR COUNT */
    STATB(0) = STATB(0) + 1;
    RETURN PORT;
  END;
END DET$ADDR$NL;

```

```

/*****
* DATE: 2 OCT 84
* VERSION: 1.1
* NAME: MOVETO$LOCAL
* MODULE NUMBER: 7.2
* DESCRIPTION: This module routes packets between
* the local host tables. The datagrams are not formatted
* as packets, but moved as datagrams. The procedure
* moves a DATAGRAM$SIZE segment starting at the location
* indicated by TABLE$PTR to the location of the next
* available byte in the designated port.
* PASSED VARIABLES: TABLE$PTR, PORT
* RETURNS: None
* GLOBAL VARIABLES USED: TX01NE, TX02NE, TX03NE, TX04NE, STATB
* LC01TX (2, 3, 4)
* GLOBAL VARIABLES CHANGED: STATB, LC01TX (2, 3, 4)
* MODULES CALLED: None
* CALLING MODULES: ROUTE$IN
* AUTHOR: C. T. Childress
* HISTORY: 1.1 C.T. Childress - 2 OCT 84 - original SBC 544
* implementation.
*****
MOVETO$LOCAL: PROCEDURE(TABLE$PTR, PORT);

```

```

  DECLARE PORT BYTE,
    TABLE$PTR ADDRESS;

  IF (PORT >= 1 AND PORT <= 4) THEN
    DO;
      DO CASE PORT;
        ; /* CASE ZERO IS NULL */
        CALL MOVE(DATA$GRAM$SIZE, TABLE$PTR, .LC01TX(TX01NE));
      END CASE;
    END;
  END;

```

```

SOURCE CODE FOR OP544.SRC,                25 SEP 85

CALL MOVE(DATA$GRAM$SIZE, TABLE$PTR, .LC02TX(TX02NE));
CALL MOVE(DATA$GRAM$SIZE, TABLE$PTR, .LC03TX(TX03NE));
CALL MOVE(DATA$GRAM$SIZE, TABLE$PTR, .LC04TX(TX04NE));

END; /* END CASE */
CALL LD$TAB$H$SKP(PORT + 4);
END;
ELSE DO;
  STATTB(L$RIS$DEST$ERR) = STATTB(L$RIS$DEST$ERR) + 1;
  STATTB(0) = STATTB(0) + 1;
END;

```

```

END MOVETO$LOCAL;

```

```

/***** 2 OCT 84 *****/
* DATE: 2 OCT 84
* VERSION: 1.1
* NAME: ROUTE$IN
* MODULE NUMBER: 7.0
* DESCRIPTION: This module routes datagrams from the
  local receive tables to local transmit tables, if
  destined for another local host, or to the datalink
  layer if destined for another UNID. The address byte
  is evaluated for the destination processing. This
  procedure along with ROUTE$OUT is called in and endless
  loop by the main program.
* PASSED VARIABLES: None
* RETURNS: None
* GLOBAL VARIABLES USED: LCOINS, LC02NS, LC03NS, LC04NS,
  DESTINATION, STATTB, LSEM$1, LSEM$2, LSEM$3,
  LSEM$4, NSEM$1, NSEM$2, DESTINATION$ADDRESS
* GLOBAL VARIABLES CHANGED: All of those above
* MODULES CALLED: SENDSEQ, SRVC$TAB$H$SKP, DET$ADDR,
  MOVETO$LOCAL, SEND$PACKET, DET$ADDR$NL
* CALLING MODULES:
* AUTHOR: C. T. Childress
* HISTORY: 1.1 C.T. Childress - 2 OCT 84 - original SBC 544
  implementation.
*****/
ROUTE$IN: PROCEDURE;
DECLARE NETR1 BASED NPTR$1 (1) BYTE;
DECLARE NETR2 BASED NPTR$2 (1) BYTE;
/* NETRX POINTS TO A PACKET SIZED ENTRY */

```

```

IF (((LCOINE - LCOINS) >= DATA$GRAM$SIZE) OR (LCOINS > LCOINE)) THEN
DO;

```

```

CALL DET$ADDR(.LC01RX(LC01NS + OFFSET1));
IF DESTINATION = 0 THEN
DO;
CALL MOVETO$LOCAL(.LC01RX(LC01NS + OFFSET1), (DESTINATION$ADDRESS AND OFH));
END;
ELSE
IF DESTINATION = 1 THEN
DO;
IF LSEM$1 = DONE THEN
CALL SEND$PACKET(.LC01RX(LC01NS), 1);
END;
ELSE
DO;
STATTB(L$RI$DEST$ERR) = STATTB(L$RI$DEST$ERR) + 1;
END;
CALL SRVC$TAB$H$SKP(1);
END;

IF (((LC02NE - LC02NS) >= DATA$GRAM$SIZE) OR (LC02NS > LC02NE)) THEN
DO;
CALL DET$ADDR(.LC02RX(LC02NS + OFFSET1));
IF DESTINATION = 0 THEN
DO;
CALL MOVETO$LOCAL(.LC02RX(LC02NS + OFFSET1), (DESTINATION$ADDRESS AND OFH));
END;
ELSE
IF DESTINATION = 1 THEN
DO;
IF LSEM$2 = DONE THEN
CALL SEND$PACKET(.LC02RX(LC02NS), 2);
END;
ELSE
DO;
STATTB(L$RI$DEST$ERR) = STATTB(L$RI$DEST$ERR) + 1;
END;
CALL SRVC$TAB$H$SKP(2);
END;

IF (((LC03NE - LC03NS) >= DATA$GRAM$SIZE) OR (LC03NS > LC03NE)) THEN
DO;
CALL DET$ADDR(.LC03RX(LC03NS + OFFSET1));
IF DESTINATION = 0 THEN
DO;
CALL MOVETO$LOCAL(.LC03RX(LC03NS + OFFSET1), (DESTINATION$ADDRESS AND OFH));
END;
ELSE
IF DESTINATION = 1 THEN
DO;
IF LSEM$3 = DONE THEN
CALL SEND$PACKET(.LC03RX(LC03NS), 3);

```

```

      END;
    ELSE
      DO:
        STATTB(L$RI$DEST$ERR) = STATTB(L$RI$DEST$ERR) + 1;
      END;
      CALL SRVC$TAB$H$SKP(3);
    END;

    IF (((LC04NE - LC04NS) >= DATA$GRAM$SIZE) OR (LC04NS > LC04NE)) THEN
      DO:
        CALL DET$ADDR(.LC04RX(LC04NS + OFFSET1));
        IF DESTINATION = 0 THEN
          DO:
            CALL MOVETO$LOCAL(.LC04RX(LC04NS + OFFSET1), (DESTINATION$ADDRESS AND OFH));
          END;
        ELSE
          IF DESTINATION = 1 THEN
            DO:
              IF LSEM$4 = DONE THEN
                CALL SEND$PACKET(.LC04RX(LC04NS), 4);
              END;
            ELSE
              DO:
                STATTB(L$RI$DEST$ERR) = STATTB(L$RI$DEST$ERR) + 1;
              END;
              CALL SRVC$TAB$H$SKP(4);
            END;
          END;
        IF NSEM$1 = READY THEN
          DO:
            DESTINATION$ADDRESS = DET$ADDR$NL(.NETR1(0));
            IF DESTINATION$ADDRESS >= 0 THEN
              CALL MOVETO$LOCAL(.NETR1(OFFSET1), DESTINATION$ADDRESS);
              NSEM$1 = DONE;
              CALL SRVC$TAB$H$SKP(9); /* FOR SIMULATION ONLY */
            END;
          END;
        IF NSEM$2 = READY THEN
          DO:
            DESTINATION$ADDRESS = DET$ADDR$NL(.NETR2(0));
            IF DESTINATION$ADDRESS >= 0 THEN
              CALL MOVETO$LOCAL(.NETR2(OFFSET1), DESTINATION$ADDRESS);
              NSEM$2 = DONE;
              CALL SRVC$TAB$H$SKP(10); /* FOR SIMULATION ONLY */
            END;
          END;
        END ROUTE$IN;
      /*****

```

SOURCE CODE FOR OP544.SRC,

25 SEP 85

```

* DATE: 25 SEP 85
* VERSION: 1.2
* NAME: ROUTESOUT
* MODULE NUMBER: 8.0
* DESCRIPTION: This module routes datagrams from the
* local transmit tables to the local hosts. Version 1.2
* removes the interrupt driven transmit procedures and
* replaces them with polled routines. This procedure
* along with ROUTESIN is called in an endless loop by
* the main program.
*
* PASSED VARIABLES: None
* RETURNS: None
* GLOBAL VARIABLES USED: TRTA$1 (2,3,4), TXTR$1 (2,3,4),
* SEND$1 (2,3,4), TX01NE (2,3,4), TX01NS (2,3,4)
* GLOBAL VARIABLES CHANGED: All of those above
* MODULES CALLED: SENDSEQ, SERVICE$TRANS$1 (2, 3, 4)
* CALLING MODULES: main
* AUTHOR: Mark Weber
* HISTORY: 1.2 Mark Weber - 25 SEP 85 - implemented polled transmit
* 1.1 C.T. Childress - 2 OCT 84 - original SBC 544
* implementation.
*****/
ROUTESOUT: PROCEDURE;
IF (((TX01NE - TX01NS) >= DATA$GRAM$SIZE) OR (TX01NS > TX01NE)) AND
(NOT SEND$1) THEN
DO;
CALL SNDSEQ(.TP$2, SIZE(TP$2));
IF (TRTA$1 AND ((NOT TXTR$1) AND (NOT RXTA$1))) AND
((NOT RXTR$1) AND (NOT TXTA$1))) THEN
DO;
CALL SNDSEQ(.TP$3, SIZE(TP$3));
TXTR$1 = TRUE;
SEND$1 = TRUE;
OUTPUT(US$PO$DATA) = TR;
END;
IF ((NOT TRTA$1) OR ((TXTR$1 AND RXTA$1) AND
((NOT RXTR$1) AND (NOT TXTA$1)))) THEN
DO;
CALL SNDSEQ(.TP$4, SIZE(TP$4));
SEND$1 = TRUE;
CALL SERVICE$TRANS$1;
END;
END;

IF (((TX02NE - TX02NS) >= DATA$GRAM$SIZE) OR (TX02NS > TX02NE)) AND
(NOT SEND$2) THEN
DO;

```

```
CALL SNDSEQ(.TP$2, SIZE(TP$2));
IF (TRTA$2 AND ((NOT TXTR$2) AND (NOT RXTA$2)) AND
    ((NOT RXTR$2) AND (NOT TXTA$2))) THEN
```

```
DO;
CALL SNDSEQ(.TP$3, SIZE(TP$3));
TXTR$2 = TRUE;
SEND$2 = TRUE;
OUTPUT(US$P1$DATA) = TR;
END;
```

```
IF ((NOT TRTA$2) OR ((TXTR$2 AND RXTA$2) AND
    ((NOT RXTR$2) AND (NOT TXTA$2)))) THEN
```

```
DO;
CALL SNDSEQ(('.', TO TRANS$2, 12);
CALL SNDSEQ(.TP$4, SIZE(TP$4));
SEND$2 = TRUE;
CALL SERVICE$TRANS$2;
```

```
END;
```

```
IF (((TX03NE - TX03NS) >= DATA$GRAM$SIZE) OR (TX03NS > TX03NE)) AND
    (NOT SEND$3) THEN
```

```
DO;
```

```
CALL SNDSEQ(.TP$2, SIZE(TP$2));
```

```
IF (TRTA$3 AND ((NOT TXTR$3) AND (NOT RXTA$3)) AND
    ((NOT RXTR$3) AND (NOT TXTA$3))) THEN
```

```
DO;
```

```
CALL SNDSEQ(.TP$3, SIZE(TP$3));
```

```
TXTR$3 = TRUE;
```

```
SEND$3 = TRUE;
```

```
OUTPUT(US$P2$DATA) = TR;
```

```
END;
```

```
IF ((NOT TRTA$3) OR ((TXTR$3 AND RXTA$3) AND
    ((NOT RXTR$3) AND (NOT TXTA$3)))) THEN
```

```
DO;
```

```
CALL SNDSEQ(('.', TO TRANS$3, 12);
```

```
CALL SNDSEQ(.TP$4, SIZE(TP$4));
```

```
SEND$3 = TRUE;
```

```
CALL SERVICE$TRANS$3;
```

```
END;
```

```
IF (((TX04NE - TX04NS) >= DATA$GRAM$SIZE) OR (TX04NS > TX04NE)) AND
    (NOT SEND$4) THEN
```

```
DO;
```

```
CALL SNDSEQ(.TP$2, SIZE(TP$2));
```

```
IF (TRTA$4 AND ((NOT TXTR$4) AND (NOT RXTA$4)) AND
    ((NOT RXTR$4) AND (NOT TXTA$4))) THEN
```

```
DO;
```

25 SEP 85

SOURCE CODE FOR OP544.SRC.

```

CALL SNDSEQ(.TP$3, SIZE(TP$3));
TXTR$4 = TRUE;
SEND$4 = TRUE;
OUTPUT(US$P3$DATA) = TR;
END;
IF ((NOT TRTA$4) OR ((TXTR$4 AND RXTA$4) AND
  ((NOT RXTR$4) AND (NOT TXTA$4)))) THEN
DO;
CALL SNDSEQ(.(' TO TRANS$4.', 12);
CALL SNDSEQ(.TP$4, SIZE(TP$4));
SEND$4 = TRUE;
CALL SERVICE$TRANS$4;
END;
END;

```

END ROUTE\$OUT;

```

/*****
* DATE: 2 OCT 84
* VERSION: 1.1
* NAME: SERVICE$LOOP
* MODULE NUMBER: 9.1
* DESCRIPTION: This module routes a frame back to the
* network layer software. The source and destination
* addresses are switched. This procedure simulates only
* one datalink channel ( channel 1 ).
* PASSED VARIABLES: TABLE$PTR, PORT
* RETURNS: None
* GLOBAL VARIABLES USED: NTOINE, LSEM$1 (2, 3, 4), NPTR$1, NSEM1
* GLOBAL VARIABLES CHANGED: All of those above
* MODULES CALLED: LD$TAB$H$SKP
* CALLING MODULES: LOOP
* AUTHOR: C. T. Childress
* HISTORY: 1.1 C.T. Childress - 2 OCT 84 - original SBC 544
* implementation.
*****
SERVICE$LOOP: PROCEDURE(TABLE$PTR, PORT);
DECLARE
  PORT BYTE,
  TABLE$PTR ADDRESS,
  LCOXR$ BASED TABLE$PTR (1) BYTE;

DO INDEX = 0 TO (PACKET$SIZE - 1); /* SWAP DATA */
  NTOIRX(NTOINE + INDEX + 2) = LCOXR$(INDEX);
END;
NTOIRX(NTOINE + 0) = 0;
NTOIRX(NTOINE + 1) = 0;

```



```

NT01RX(NT01NE + 3) = ROR(NT01RX(NT01NE + 3), 4); /* SWAP SRC/DEST */
NT01RX(NT01NE + 6) = LCOXR(5);
NT01RX(NT01NE + 7) = LCOXR(4); /* SWAP PACKET HEADER */

```

```

DO INDEX = 0 TO 3; /* SWAP IP DEST & SOURCE */
  NT01RX(NT01NE + INDEX + 24) = LCOXR(INDEX + 26);
  NT01RX(NT01NE + INDEX + 28) = LCOXR(INDEX + 22);
END;

```

```

CALL LD$TAB$H$SKP(9);
DO CASE PORT;

```

```

  LSEM$1 = DONE;
  LSEM$2 = DONE;
  LSEM$3 = DONE;
  LSEM$4 = DONE;

```

```

END;
NPTR$1 = .NT01RX(NT01NS + 2);
NSEM$1 = READY;

```

```

END SERVICE$LOOP;

```

```

/*****
* DATE: 2 OCT 84
* VERSION: 1.1
* NAME: LOOP
* MODULE NUMBER: 9.0
* DESCRIPTION: This module simulates the operation
  of the SBC 88/45. Semaphores and pointers are set
  and the frame is then looped back to the sender.
  This routine operates only on datalink channel 1 and
  the frame must be destined for the same UNID and
  CHANNEL as the was the source.
* PASSED VARIABLES: None
* RETURNS: None
* GLOBAL VARIABLES USED: LSEM$1 (2, 3, 4), NSEM$1,
  SEND$1 (2, 3, 4)
* GLOBAL VARIABLES CHANGED: All of those above
* MODULES CALLED: SERVICE$LOOP
* CALLING MODULES: main
* AUTHOR: C. T. Childress
* HISTORY: 1.1 C.T. Childress - 2 OCT 84 - original SBC 544
  implementation.
*****/
LOOP: PROCEDURE; /* NSEM$1 IS USED FOR THE SIMULATION AND SHOULD NOT
  BE USED FOR THE 88/45 OPERATIONAL SOFTWARE. THE 88/45
  NEEDS ONLY TO INTERROGATE THE LSEM$X VARIABLE. */

```

SOURCE CODE FOR OP544.SRC,

25 SEP 85

```

IF (LSEM$1 = READY AND NSEM$1 = DONE) AND (NOT SEND$1) THEN
  CALL SERVICE$LOOP(LPTR$1, 1);
IF (LSEM$2 = READY AND NSEM$1 = DONE) AND (NOT SEND$2) THEN
  CALL SERVICE$LOOP(LPTR$2, 2);
IF (LSEM$3 = READY AND NSEM$1 = DONE) AND (NOT SEND$3) THEN
  CALL SERVICE$LOOP(LPTR$3, 3);
IF (LSEM$4 = READY AND NSEM$1 = DONE) AND (NOT SEND$4) THEN
  CALL SERVICE$LOOP(LPTR$4, 4);

```

END LOOP;

```

/*****
* DATE: 2 OCT 84
* VERSION: 1.1
* NAME: SET$TRTA
* MODULE NUMBER: 7.0
* DESCRIPTION: This module performs the TRTA
  handshake reinitialization for each loop in the main
  program. This procedure is called in an endless
  loop by the main program.
* PASSED VARIABLES: None
* RETURNS: None
* GLOBAL VARIABLES USED: TRTA$1 (2, 3, 4), CHAR$X
* GLOBAL VARIABLES CHANGED: All of those above
* MODULES CALLED: SNDSEQ, RXINS1
* CALLING MODULES: main
* AUTHOR: C. T. Childress
* HISTORY: 1.1 C.T. Childress - 2 OCT 84 - original SBC 544
  implementation.
*****
SET$TRTA: PROCEDURE;

```

```

CALL SNDSEQ(.MSG1, SIZE(MSG1));
HALT;
DO WHILE ((CHAR$X = 'V') OR (CHAR$X = 'Y'));
  IF ((CHAR$X = 'V') OR (CHAR$X = 'Y')) THEN
    DO;
      CALL SNDSEQ(.MSG2, SIZE(MSG2));
      HALT;
      CHAR$X = CHAR$X - 31H;
      IF ((CHAR$X >= 0) AND (CHAR$X <= 3)) THEN
        DO CASE CHAR$X;
          TRTA$1 = TRUE;
          TRTA$2 = TRUE;
          TRTA$3 = TRUE;

```

SOURCE CODE FOR OP544.SRC, 25 SEP 85

```

SOURCE CODE FOR OP544.SRC, 25 SEP 85

        TRTA$4 = TRUE;
END;
ELSE
    TRTA$1, TRTA$2, TRTA$3, TRTA$4 = FALSE;
END;
CALL SNDSEQ(.MSG1, SIZE(MSG1));
HALT;
END;

END SET$TRTA;

/***** THIS IS THE MAIN BODY OF THE PROGRAM *****/
/***** THIS IS THE MAIN PROGRAM MODULE. *****/
/***** THE ROUTE$IN AND ROUTE$OUT PROCEDURES. THIS ALLOWS *****/
/***** THE SBC544 TO PROCESS LOCAL HOST MESSAGES. AN *****/
/***** ADDITIONAL CALL TO LOOP SERVES TO PROVIDE THE *****/
/***** DUMMY 88/45 INTERFACE PROCEDURES. WHEN CONFIGURED *****/
/***** WITH AN OPERATION SBC 88/45, THE CALL TO LOOP MUST BE *****/
/***** REMOVED. *****/
/***** PASSED VARIABLES: *****/
/***** RETURNS: *****/
/***** GLOBAL VARIABLES USED: *****/
/***** GLOBAL VARIABLES CHANGED: *****/
/***** MODULES CALLED: *****/
/***** CALLING MODULES: *****/
/***** AUTHOR: C. T. Childress *****/
/***** HISTORY: 1.1 C.T. Childress - 2 OCT 84 - original SBC 544 *****/
/***** implementation. *****/
/***** *****/

BEGIN:
DISABLE;
CALL $SMASK($SIMSMASK);
CALL INITIALIZE$BOARD;
CALL SNDSEQ(.HEADER, SIZE(HEADER));
CALL SNDSEQ(.TP$5, SIZE(TP$5));
CALL INIT;
CALL SNDSEQ(.TP$6, SIZE(TP$6));

```

25 SEP 85

SOURCE CODE FOR OP544.SRC,

```
CALL INIT$TAB;  
CALL SNDSEQ(.TP$7, SIZE(TP$7));  
ENABLE;
```

```
CALL SET$TRTA;  
FOREVER = TRUE;  
CALL SNDSEQ(.TP$8, SIZE(TP$8));  
DO WHILE FOREVER;  
  CALL ROUTE$IN;  
  CALL ROUTE$OUT;  
  CALL LOOP;  
END;
```

END MAIN;

/***** THE END *****/

INTR3.LOC

The following listing is the include file used with the source code for OP544.SRC. This include file contains the modified I/O procedures for the polled transmit ports of the SBC 544.

INCLUDE SOURCE CODE FOR OPS44.SRC (INTR3.LOC)

```

/*****
* DATE: 25 SEP 85
* VERSION: 1.2
* NAME: RXIN$1
* MODULE NUMBER: 7.6
* DESCRIPTION: This procedure is used with the
* SET$TRTA routine. This procedure is used when channel
* one of the SBC544 must serve as a monitor and read input
* values from the terminal.
* PASSED VARIABLES: None
* RETURNS: None
* GLOBAL VARIABLES USED: CHAR$X
* GLOBAL VARIABLES CHANGED: CHAR$X
* MODULES CALLED: None
* CALLING MODULES: SET$TRTA
* AUTHOR: Mark Weber
* HISTORY: 1.1 C.T. Childress - 2 OCT 84 - original SBC 544
* implementation.
*****/
RXIN$1: PROCEDURE INTERRUPT 8;

```

```

CHAR$X = INPUT(US$PO$DATA);
OUTPUT(US$PO$DATA) = CHAR$X;
OUTPUT(IC$PORTA) = IC$EOI;
END RXIN$1;

```

```

/*****
* DATE: 25 SEP 85
* VERSION: 1.2
* NAME: TXOUT$1
* MODULE NUMBER: 7.6
* DESCRIPTION: This procedure is polled by the
* SET$TRTA routine. This procedure is used when channel
* one of the SBC544 must serve as a monitor and read input
* values from the terminal.
* PASSED VARIABLES: None
* RETURNS: None
* GLOBAL VARIABLES USED: None
* GLOBAL VARIABLES CHANGED: None
* MODULES CALLED: None
* CALLING MODULES: None
* AUTHOR: Mark Weber
* HISTORY: 1.2 Mark Weber - 25 SEP 85 - changed to polled routine
* 1.1 C.T. Childress - 2 OCT 84 - original SBC 544
* implementation.
*****/
/* THIS PROCEDURE IS USED WHEN THE LOCAL CHANNEL SERVES ONLY AS A MONITOR
TXOUT$1: PROCEDURE ;
OUTPUT(IC$PORTA) = IC$EOI;

```

```

INCLUDE SOURCE CODE FOR OP544.SRC (INTR3.LOC)

END TXOUT$1;
/*
** *****
** DATE: 26 SEP 86
** VERSION: 1.2
** NAME: SERVICE$RCV$1
** MODULE NUMBER: 7.6
** DESCRIPTION: This procedure is interrupt driven and
** operates at all times once initialized. The procedure
** checks for valid handshakes between the host and UND,
** then receives one byte of data at a time. The datagram
** is loaded into the receive table of PACKET$TABLE$SIZE
** offset by the amount OFFSET$1. Procedure SERVICE$RCV$1
** was added to the program in version 1.2.
**
** PASSED VARIABLES: None
** RETURNS: None
** GLOBAL VARIABLES USED: SEMD$1 TXTR$1, RXTR$1, RXTA$1,
** LCOINE, BYTES$RCV$1, TXTR$1
** GLOBAL VARIABLES CHANGED: All of those above
** MODULES CALLED: SNDSEQ,
** CALLING MODULES: NONE (interrupt driven)
** AUTHOR: Mark Weber
** HISTORY: 1.2 Mark Weber - 25 SEP 85 - added SERVICE$RCV$1
** 1.1 C.T. Childress - 2 OCT 84 - original SBC 544
** *****
** *****
SERVICE$RCV$1: PROCEDURE INTERRUPT 8;
/*
CHAR$2 = INPUT(US$POS$DATA);
IF ((NOT TRTA$1) OR ((RXTR$1 AND TXTR$1) AND
((NOT TXTR$1) AND (NOT RXTA$1)))) THEN
DO;
LCO1RX(LCOINE + OFFSET1) = CHAR$1;
BYTES$RCV$1 = BYTES$RCV$1 + 1;
LCOINE = LCOINE + 1;
IF BYTES$RCV$1 >= DATA$GRAM$SIZE THEN
DO;
LCOINE = LCOINE + OFFSET1;
IF(LCOINE >= LCO1S2) THEN
LCOINE = 0;
BYTES$RCV$1 = 0;
RXTR$1 = FALSE;
TXTR$1 = FALSE;
SEMD$1 = FALSE;
END;
END;
IF TRTA$1 THEN

```

INCLUDE SOURCE CODE FOR OPS44.SRC (INTR3.LOC)

```

DO:
  IF CHAR$1 = TA THEN
    IF ((TXTR$1 AND (NOT RXTA$1)) AND
       ((NOT RXTR$1) AND (NOT TTXA$1))) THEN
      DO:
        RXTA$1 = TRUE;
        SEND$1 = FALSE;
        CALL SNDSEQ(.TP$3C, SIZE(TP$3C));
      END;
    IF CHAR$1 = TR THEN
      IF ((NOT RXTR$1) AND (NOT TTXA$1)) AND
         ((NOT TXTR$1) AND (NOT RXTA$1))) THEN
        DO:
          RXTR$1 = TRUE;
          TTXA$1 = TRUE;
          SEND$1 = TRUE;
          CALL SNDSEQ(.TP$3B, SIZE(TP$3B));
          CALL SNDSEQ(.TP$3A, SIZE(TP$3A));
          OUTPUT(US$PO$DATA) = TA;
        END;
      END;
    END;
  
```

OUTPUT(IC\$PORTA) = IC\$EOI;

END SERVICE\$RCV\$1;

```

*/
/*****
* DATE: 2 OCT 84
* VERSION: 1.1
* NAME: SERVICE$RCV$2
* MODULE NUMBER: 7.7
* DESCRIPTION: This procedure is interrupt driven and
* operates at all times once initialized. The procedure
* checks for valid handshakes between the host and UND.
* then receives one byte of data at a time. The datagram
* is loaded into the receive table of PACKET$TABLE$SIZE
* offset by the amount OFFSET$1.
* PASSED VARIABLES: None
* RETURNS: None
* GLOBAL VARIABLES USED: LC02NE, BYTES$RECV$2, RXTR$2, RXTA$2,
* SEMD$2, TTXA$2, TXTR$2
* GLOBAL VARIABLES CHANGED: All of those above
* MODULES CALLED: SNDSEQ,
* CALLING MODULES: NONE (interrupt driven)
* AUTHOR: Mark Weber
* HISTORY: 1.1 C.T. Childress - 2 OCT 84 - original SBC 544
* implementation.
* *****/
SERVICE$RCV$2: PROCEDURE INTERRUPT 10;
  
```



```

INCLUDE SOURCE CODE FOR OP544.SRC (INTR3.LOC)

CHAR$2 = INPUT(USP1$DATA);

IF ((NOT TRTA$2) OR ((RXTR$2 AND TXTA$2) AND
  ((NOT RXTR$2) AND (NOT RXTA$2)))) THEN
DO;
  LC02RX(LC02NE + OFFSET1) = CHAR$2;
  BYTES$RCV$2 = BYTES$RCV$2 + 1;
  LC02NE = LC02NE + 1;
  IF BYTES$RCV$2 >= DATA$GRAM$SIZE THEN
  DO;
    LC02NE = LC02NE + OFFSET1;
    IF(LC02NE >= LC02SZ) THEN
      LC02NE = 0;
    BYTES$RCV$2 = 0;
    RXTR$2 = FALSE;
    TXTA$2 = FALSE;
    SEND$2 = FALSE;
  END;
END;

IF TRTA$2 THEN
DO;
  IF CHAR$2 = TA THEN
  IF ((TXTR$2 AND (NOT RXTA$2)) AND
    ((NOT RXTR$2) AND (NOT TXTA$2))) THEN
  DO;
    RXTA$2 = TRUE;
    SEND$2 = FALSE;
    CALL SNOSEQ(.TP$3C, SIZE(TP$3C));
  END;
  IF CHAR$2 = TR THEN
  IF ((NOT RXTR$2) AND (NOT TXTA$2)) AND
    ((NOT TXTR$2) AND (NOT RXTA$2))) THEN
  DO;
    RXTR$2 = TRUE;
    TXTA$2 = TRUE;
    SEND$2 = TRUE;
    CALL SNOSEQ(.TP$3B, SIZE(TP$3B));
    CALL SNOSEQ(.TP$3A, SIZE(TP$3A));
    OUTPUT(USP1$DATA) = TA;
  END;
END;

OUTPUT(IC$PORTA) = IC$EOI;

END SERVICE$RCV$2;

```

```

/*****
* DATE: 2 OCT 84
*

```

INCLUDE SOURCE CODE FOR OP544.SRC (INTR3.LOC)

```

* VERSION:
* NAME:
* MODULE NUMBER:
* DESCRIPTION:
* operates at all times once initialized. The procedure
* checks for valid handshakes between the host and UND,
* then receives one byte of data at a time. The datagram
* is loaded into the receive table of PACKET$TABLE$SIZE
* offset by the amount OFFSET$1.
* PASSED VARIABLES:
* RETURNS:
* GLOBAL VARIABLES USED: SEMD$3 TXTR$3, RXTR$3, RXTA$3,
* GLOBAL VARIABLES CHANGED: All of those above
* MODULES CALLED: SNOSEQ,
* CALLING MODULES: NONE (interrupt driven)
* AUTHOR: Mark Weber
* HISTORY:
* 1.1 C.T. Childress - 2 OCT 84 - original SBC 544
* implementation.
* *****/
SERVICE$RCV$3: PROCEDURE INTERRUPT 12;
CHAR$3 = INPUT(US$P2$DATA);
IF ((NOT TRTA$3) OR ((RXTR$3 AND TXTR$3) AND
((NOT TXTR$3) AND (NOT RXTA$3)))) THEN
DO;
LC03RX(LC03NE + OFFSET1) = CHAR$3;
BYTES$RECV$3 = BYTES$RECV$3 + 1;
LC03NE = LC03NE + 1;
IF BYTES$RECV$3 >= DATA$GRAM$SIZE THEN
DO;
LC03NE = LC03NE + OFFSET1;
IF (LC03NE >= LC03S2) THEN
LC03NE = 0;
BYTES$RECV$3 = 0;
RXTR$3 = FALSE;
TXTR$3 = FALSE;
SEMD$3 = FALSE;
END;
END;
IF TRTA$3 THEN
DO;
IF CHAR$3 = TA THEN
IF ((TXTR$3 AND (NOT RXTR$3)) AND
((NOT RXTR$3) AND (NOT TXTR$3))) THEN
DO;
RXTA$3 = TRUE;

```

```
INCLUDE SOURCE CODE FOR OP544.SRC (INTR3.LOC)
```

```
SEND$3 = FALSE;
CALL SNDSEQ(.TP$3C, SIZE(TP$3C));
END;

IF CHAR$3 = TR THEN
  IF ((NOT RXTR$3) AND (NOT TXTA$3)) AND
     ((NOT TXTR$3) AND (NOT RXTA$3))) THEN
    DO;
      RXTR$3 = TRUE;
      TXTA$3 = TRUE;
      SEND$3 = TRUE;
      CALL SNDSEQ(.TP$3B, SIZE(TP$3B));
      CALL SNDSEQ(.TP$3A, SIZE(TP$3A));
      OUTPUT(US$P2$DATA) = TA;
    END;
  END;
END;
```

```
OUTPUT(IC$PORTA) = IC$EOI;
END SERVICE$RCV$3;
```

```
/****** 2 OCT 84 *****
* DATE: 2 OCT 84
* VERSION: 1.1
* NAME: SERVICE$RCV$4
* MODULE NUMBER: 7.9
* DESCRIPTION: This procedure is interrupt driven and
* operates at all times once initialized. The procedure
* checks for valid handshakes between the host and UND,
* then receives one byte of data at a time. The datagram
* is loaded into the receive table of PACKET$TABLE$SIZE
* offset by the amount OFFSEI$1.
* PASSED VARIABLES: None
* RETURNS: None
* GLOBAL VARIABLES USED: LCO4NE, BYTES$RECV$4, RXTR$4, RXTA$4,
* SEMD$4, TXTA$4, TXTR$4
* GLOBAL VARIABLES CHANGED: SEMD$4, TXTA$4, TXTR$4
* MODULES CALLED: All of those above
* CALLING MODULES: SNDSEQ,
* AUTHOR: Mark Weber
* HISTORY: NONE (interrupt driven)
* 1.1 C.T. Childress - 2 OCT 84 - original SBC 544
* implementation.
*****/
SERVICE$RCV$4: PROCEDURE INTERRUPT 14;

CHAR$4 = INPUT(US$P3$DATA);

IF ((NOT TRTA$4) OR ((RXTR$4 AND TXTA$4) AND
  ((NOT TXTR$4) AND (NOT RXTA$4)))) THEN
```

INCLUDE SOURCE CODE FOR OP544.SRC (INTR3.LOC)

```

DO;
  LC04RX(LC04NE + OFFSET1) = CHAR$4;
  BYTES$RCV$4 = BYTES$RCV$4 + 1;
  LC04NE = LC04NE + 1;
  IF BYTES$RCV$4 >= DATA$GRAM$SIZE THEN
    DO;
      LC04NE = LC04NE + OFFSET1;
      IF(LC04NE >= LC04S2) THEN
        LC04NE = 0;
      BYTES$RCV$4 = 0;
      RXTR$4 = FALSE;
      TXTA$4 = FALSE;
      SEND$4 = FALSE;
    END;
  END;

  IF TRTA$4 THEN
    DO;
      IF CHAR$4 = TA THEN
        IF ((TXTR$4 AND (NOT RXTA$4)) AND
            ((NOT RXTR$4) AND (NOT TXTA$4))) THEN
          DO;
            RXTA$4 = TRUE;
            SEND$4 = FALSE;
            CALL SNDSEQ(.TP$3C, SIZE(TP$3C));
          END;
        IF CHAR$4 = TR THEN
          IF ((NOT RXTR$4) AND (NOT TXTA$4)) AND
             ((NOT TXTR$4) AND (NOT RXTA$4))) THEN
            DO;
              RXTR$4 = TRUE;
              TXTA$4 = TRUE;
              SEND$4 = TRUE;
              CALL SNDSEQ(.TP$3B, SIZE(TP$3B));
              CALL SNDSEQ(.TP$3A, SIZE(TP$3A));
              OUTPUT(USP$3$DATA) = TA;
            END;
          END;
        END;

      OUTPUT(IC$PORTA) = IC$EOI;

    END SERVICES$RCV$4;

  /*****
  /* PROCEDURE SERVICES$TRANS$1 SENDS DATA OUT CHANNEL ONE
  /*
  /*
  /* THE PURPOSE OF THIS PROCEDURE IS TO SEND A DATAGRAM OF DATA OUT
  /* LOCAL CHANNEL ONE. A SINGLE BYTE IS TRANSMITTED EACH TIME AN
  /* INTERRUPT IS GENERATED BY USART ONE ON THE TRANSMIT SIDE.
  /*
  /*
  /*****/

```

```

/* INPUT - NONE (INTERRUPT DRIVEN)
/* PROCESSING - SENDS A BYTE OF DATA FROM THE TRANSMIT ARRAY TO
/* THE DATA PORT. WHEN MESSAGE IS DONE IT RESETS
/* THE TRANSMIT INTERRUPT AND SETS TRANS$RDY TO TRUE.
/* OUTPUT - NONE.
/* INTERFACE - CALLED BY MAIN PROCEDURE.
/*
/* *****

```

```

/*****
* DATE: 25 SEP 85
* VERSION: 1.2
* NAME: SERVICE$TRAN$S1
* MODULE NUMBER: 8.4
* DESCRIPTION:
    This procedure is a polled transmit
    service routine that is called in an endless loop
    through procedure ROUTE$OUT. The procedure transmits
    a DATAGRAM$SIZE segment of data in the local transmit
    table. The receive interrupts stay active during the
    operation of this procedure.
* PASSED VARIABLES: None
* RETURNS: None
* GLOBAL VARIABLES USED: LCO$INE, BYTES$SENT$1, TXTR$1, TXTA$1,
    SEMD$1 RXTA$1, RXTR$1
* GLOBAL VARIABLES CHANGED: All of those above
* MODULES CALLED: SNDSQC,
    ROUTE$OUT
* CALLING MODULES:
* AUTHOR: Mark Weber
* HISTORY: 1.2 Mark Weber - 25 SEP 85 - polled transmit routines
    1.1 C.T. Childress - 2 OCT 84 - original SBC 544
    implementation.
*****/
SERVICE$TRAN$S1: PROCEDURE ;

```

```

DO:
DO WHILE BYTESSENT$1 < DATA$GRAM$SIZE:
DO WHILE NOT (INPUT(US$POSTAT) AND 1): END:
OUTPUT(US$POSDATA) = LCOITX(TXOINS):
BYTESSENT$1 = BYTESSENT$1 + 1:
TXOINS = TXOINS + 1:
END:
IF BYTESSENT$1 >= DATA$GRAM$SIZE THEN
DO:
BYTESSENT$1 = 0:
IF TXOINS >= TXO2SZ THEN
TXOINS = 0:
TXTR$1 = FALSE:
RXTA$1 = FALSE:
SEND$1 = FALSE:
CALL SNDSEQ(.TP$4A, SIZE(TP$4A)):

```

END;
END;

END SERVICE\$TRANS\$1;

```

/*****
** DATE:                25 SEP 85
** VERSION:             1.2
** NAME:                SERVICE$TRANSMIT
** MODULE NUMBER:      8.5
** DESCRIPTION:         This procedure is a polled transmit
                        service routine that is called in an endless loop
                        through procedure ROUTE$OUT. The procedure transmits
                        a DATAGRAM$SIZE segment of data in the local transmit
                        table. The receive interrupts stay active during the
                        operation of this procedure.
** PASSED VARIABLES:    None
** RETURNS:            None
** GLOBAL VARIABLES USED: LCOZNE, BYTES$SENT$, TXTR$, TXTAS$,
                        SEMD$, RXTA$, RXTR$
** GLOBAL VARIABLES CHANGED: SEMD$
** MODULES CALLED:     All of those above
** CALLING MODULES:    SNDSEQ,
                        ROUTE$OUT
** AUTHOR:              Mark Weber
** HISTORY:             1.2 Mark Weber - 25 SEP 85 - polled transmit routines
                        implementation.
                        1.1 C.T. Childress - 2 OCT 84 - original SBC 544
** *****/
SERVICE$TRANSMIT: PROCEDURE ;

```

```

DO: DO WHILE BYTES$SENT$2 < DATA$GRAM$SIZE;
      DO WHILE NOT (INPUT(US$P1$STAT) AND 1); END;
      OUTPUT(US$P1$DATA) = LC02TX(TX02NS);
      BYTES$SENT$2 = BYTES$SENT$2 + 1;
      TX02NS = TX02NS + 1;
    END;
  IF BYTES$SENT$2 >= DATA$GRAM$SIZE THEN
    DO:
      BYTES$SENT$2 = 0;
      IF TX02NS >= TX02SZ THEN
        TX02NS = 0;
        TXTR$2 = FALSE;
        RXTA$2 = FALSE;
        SEND$2 = FALSE;
        CALL SNDSEQ(.TP$4A, SIZE(TP$4A));
      END;
    END;
  END;
END;

```

INCLUDE SOURCE CODE FOR OPS44.SRC (INTR3.LOC)

PAGE 11

END SERVICE\$TRANS\$2:

```

/*****
* DATE: 25 SEP 85
* VERSION: 1.2
* NAME: SERVICE$TRANS$3
* MODULE NUMBER: 8.6
* DESCRIPTION: This procedure is a polled transmit
* service routine that is called in an endless loop
* through procedure ROUTE$OUT. The procedure transmits
* a DATA$GRAM$SIZE segment of data in the local transmit
* table. The receive interrupts stay active during the
* operation of this procedure.
* PASSED VARIABLES: None
* RETURNS: None
* GLOBAL VARIABLES USED: LCO3NE, BYTES$SENT$3, TXTR$3, TXTA$3,
* SEMD$3, RXTA$3, RXTR$3
* GLOBAL VARIABLES CHANGED: All of those above
* MODULES CALLED: SNDSEQ, ROUTE$OUT
* CALLING MODULES:
* AUTHOR: Mark Weber
* HISTORY: 1.2 Mark Weber - 25 SEP 85 - polled transmit routines
* 1.1 C.T. Childress - 2 OCT 84 - original SBC 544
* implementation.
*****/
SERVICE$TRANS$3: PROCEDURE :

```

```

DO:
DO WHILE BYTES$SENT$3 < DATA$GRAM$SIZE:
DO WHILE NOT (INPUT(US$P2$STAT) AND 1): END:
OUTPUT(US$P2$DATA) = LCO3TX(TXO3NS):
BYTES$SENT$3 = BYTES$SENT$3 + 1:
TXO3NS = TXO3NS + 1:
END:
IF BYTES$SENT$3 >= DATA$GRAM$SIZE THEN
DO:
BYTES$SENT$3 = 0:
IF TXO3NS >= TXO3SZ THEN
TXO3NS = 0:
TXTR$3 = FALSE:
RXTR$3 = FALSE:
SEMD$3 = FALSE:
CALL SNDSEQ(.TP$4A, SIZE(TP$4A)):
END:
END:
END SERVICE$TRANS$3:

```

INCLUDE SOURCE CODE FOR OP544.SRC (INTR3.LOC)

```

/*****
* DATE: 25 SEP 85
* VERSION: 1.2
* NAME: SERVICE$TRANS$4
* MODULE NUMBER: 8.7
* DESCRIPTION: This procedure is a polled transmit
* service routine that is called in an endless loop
* through procedure ROUTE$OUT. The procedure transmits
* a DATAGRAM$SIZE segment of data in the local transmit
* table. The receive interrupts stay active during the
* operation of this procedure.
* PASSED VARIABLES: None
* RETURNS: None
* GLOBAL VARIABLES USED: LCO4NE, BYTES$SENT$4, TXTR$4, TXTA$4,
* SEMD$4, RXTA$4, RXTR$4
* GLOBAL VARIABLES CHANGED: All of those above
* MODULES CALLED: SNDSEQ.
* CALLING MODULES: ROUTE$OUT
* AUTHOR: Mark Weber
* HISTORY: 1.2 Mark Weber - 25 SEP 85 - polled transmit routines
* 1.1 C.T. Childress - 2 OCT 84 - original SBC 544
* implementation.
*****/
SERVICE$TRANS$4: PROCEDURE ;
DO; DO WHILE BYTES$SENT$4 < DATAGRAM$SIZE;
DO WHILE NOT (INPUT(USP3$STAT) AND 1); END;
OUTPUT(USP3$DATA) = LCO4TX(TX04NS);
BYTES$SENT$4 = BYTES$SENT$4 + 1;
TX04NS = TX04NS + 1;
END;
IF BYTES$SENT$4 >= DATAGRAM$SIZE THEN
DO; BYTES$SENT$4 = 0;
IF TX04NS >= TX04SZ THEN
TX04NS = 0;
TXTR$4 = FALSE;
RXTA$4 = FALSE;
SEMD$4 = FALSE;
CALL SNDSEQ(.TP$4A, SIZE(TP$4A));
END;
END;

END SERVICE$TRANS$4;
/*****

```


APPENDIX N

DATA LINK SIMULATION SOFTWARE

1. NETX25.SRC - Main program module N - 2
2. LAPE0.SRC - Module with LAP B procedures. N - 45
3. LAPB1.SRC - Module with LAP B procedures. N - 84
4. PCKT.SRC - Module with Packet Layer procedures N - 97

```

$TITLE('UNID II NETWORK TEST PROGRAM, 18 NOV 85')
$XREF OPTIMIZE(2)
/*****
* DATE: 18 NOV 85
* VERSION: 1.2
* TITLE: ISO layer 2 LAP B simulation
* FILENAME: NETX25.SRC
* COORDINATOR: Capt Mark W. Weber
* PROJECT: UNID II
* OPERATING SYSTEM: INTEL SYSTEM III/230
* LANGUAGE: PL/M 86
* USE: This file requires no includes, but must be linked with LAPB0.OBJ,
* LAPB1.OBJ, PKT.OBJ, and SMALL.LIB. The program may be linked and
* with the SUBMIT file NETX.25.CSD.
* CONTENTS: MAIN - main processing routine
* INIT$NSTAB - initializes the necessary tables
* HEX$ASC - converts HEX to ASCII
* ASC$HEX - converts ASCII to HEX
* VALID$HEX - validates HEX numbers
* SENDSEQ - writes messages to the system console
* LD$TAB$H$SKP - updates the transmit tables
* SRVC$TAB$H$SKP - updates the receive tables
* BUILD$I$FRAME - constructs the proper I frame
* SERVICE$LOOP - sends transmitted frames to the receive table
* SERVICE$XMIT$A - dummy transmit procedure, channel A
* SERVICE$XMIT$B - dummy transmit procedure, channel B
* LOAD - loads packets to send to the datalink transmit tables
* READTAB - reads received data packets
* FIND$I$FRAMES - finds the first I frame in the table
* INITACK - resets transmit counters and timers
* ROUTE$OUT - sends frames found in the datalink transmit tables
* ROUTE$IN - processes received frames located in datalink or
* network receive tables
* START$DM$MODE - initializes the UNID for SABM operation
* READ$LINE - reads system console inputs
* START$INFO$XER - SABM mode operation
* FUNCTION: Simulates the operation of the SBC 88/45 in the UNID II using
* the datalink layer as described by CCITT recommendation X.25.
* ISIS calls are used to provide a user interface and to trace
* the process flow of the LAP B protocol employed by the CCITT
* X.25 recommendation.
* HISTORY: 1.2 Mark Weber - implemented X.25 datalink procedures
* 1.1 TRANSLATED 5 APR 84 by C.T. Chrldress
*****/

```

MAIN: DO;

```

SOURCE CODE FOR NETX25.SRC (MAIN MODULE), 18 NOV 85

/***** EXTERNAL PROCEDURES FOR ISIS SYSTEM CALLS *****/

DQ$DECODE$EXCEPTION:  PROCEDURE (ERRNUM, EXCEPTION$, STATUS) EXTERNAL;
                        DECLARE ERRNUM WORD;
                        DECLARE (EXCEPTION$, STATUS) POINTER;
                        END DQ$DECODE$EXCEPTION;

DQ$CLOSE:  PROCEDURE (AFTN, STATUS) EXTERNAL;
            DECLARE AFTN WORD, STATUS POINTER;
            END DQ$CLOSE;

DQ$DETACH:  PROCEDURE (CONNECTION, EXCEPT$) EXTERNAL;
            DECLARE CONNECTION WORD, EXCEPT$ POINTER;
            END DQ$DETACH;

DQ$EXIT:  PROCEDURE (COMPLETION$CODE) EXTERNAL;
           DECLARE COMPLETION$CODE WORD;
           END DQ$EXIT;

DQ$ATTACH:  PROCEDURE (PATH$, STATUS) WORD EXTERNAL;
            DECLARE (PATH$, STATUS) POINTER;
            END DQ$ATTACH;

DQ$CREATE:  PROCEDURE (PATH$, STATUS) WORD EXTERNAL;
            DECLARE (PATH$, STATUS) POINTER;
            END DQ$CREATE;

DQ$OPEN:  PROCEDURE (AFTN, MODE, NUM$BUF, STATUS) EXTERNAL;
           DECLARE AFTN WORD, STATUS POINTER;
           DECLARE (MODE, NUM$BUF) BYTE;
           END DQ$OPEN;

DQ$READ:  PROCEDURE (AFTN, BUFFER, COUNT, STATUS) ADDRESS EXTERNAL;
           DECLARE (AFTN, COUNT) WORD;
           DECLARE (BUFFER, STATUS) POINTER;
           END DQ$READ;

DQ$WRITE:  PROCEDURE (AFTN, BUFFER, COUNT, STATUS) EXTERNAL;
           DECLARE (AFTN, COUNT) WORD;
           DECLARE (BUFFER, STATUS) POINTER;
           END DQ$WRITE;

SND$DM:  PROCEDURE (CHANNEL, P$BIT) EXTERNAL;
         DECLARE (CHANNEL, P$BIT) BYTE;
         END SND$DM;

SND$SABM:  PROCEDURE (CHANNEL, P$BIT) EXTERNAL;
          DECLARE (CHANNEL, P$BIT) BYTE;
          END SND$SABM;

```

SOURCE CODE FOR NETX25.SRC (MAIN MODULE). 18 NOV 85

```

RCV$IN$SEQ:  PROCEDURE (CHANNEL, SEQ$NUM) BYTE EXTERNAL;
              DECLARE (CHANNEL, SEQ$NUM) BYTE;
              END RCV$IN$SEQ;

RCV$IF$FRAME: PROCEDURE (CHANNEL, SEQ$NUM) EXTERNAL;
              DECLARE (CHANNEL, SEQ$NUM) BYTE;
              END RCV$IF$FRAME;

SND$RR:      PROCEDURE (CHANNEL, P$BIT, ADDR) EXTERNAL;
              DECLARE (CHANNEL, P$BIT, ADDR) BYTE;
              END SND$RR;

UP$DATE$SEND$STATE: PROCEDURE (CHANNEL, SEQ$NUM) EXTERNAL;
                    DECLARE (CHANNEL, SEQ$NUM) BYTE;
                    END UP$DATE$SEND$STATE;

SND$REJ:     PROCEDURE (CHANNEL, P$BIT, ADDR) EXTERNAL;
              DECLARE (CHANNEL, P$BIT, ADDR) BYTE;
              END SND$REJ;

SND$RNR:     PROCEDURE (CHANNEL, P$BIT, ADDR) EXTERNAL;
              DECLARE (CHANNEL, P$BIT, ADDR) BYTE;
              END SND$RNR;

SND$UA:      PROCEDURE (CHANNEL, P$BIT) EXTERNAL;
              DECLARE (CHANNEL, P$BIT) BYTE;
              END SND$UA;

SND$DISC:    PROCEDURE (CHANNEL, P$BIT) EXTERNAL;
              DECLARE (CHANNEL, P$BIT) BYTE;
              END SND$DISC;

SND$CMDR:    PROCEDURE (CHANNEL, P$BIT, CNTL$ERROR, ERROR$STATUS) EXTERNAL;
              DECLARE (CHANNEL, P$BIT, CNTL$ERROR, ERROR$STATUS) BYTE;
              END SND$CMDR;

RCV$$SABM:   PROCEDURE (CHANNEL, P$BIT) EXTERNAL;
              DECLARE (CHANNEL, P$BIT) BYTE;
              END RCV$$SABM;

RCV$UA:      PROCEDURE (CHANNEL, F$BIT) EXTERNAL;
              DECLARE (CHANNEL, F$BIT) BYTE;
              END RCV$UA;

RCV$REJ:     PROCEDURE (CHANNEL, P$BIT, SEQ$NUM, ADDR) EXTERNAL;
              DECLARE (CHANNEL, P$BIT, SEQ$NUM, ADDR) BYTE;
              END RCV$REJ;

RCV$RNR:     PROCEDURE (CHANNEL, P$BIT, SEQ$NUM, ADDR) EXTERNAL;
              DECLARE (CHANNEL, P$BIT, SEQ$NUM, ADDR) BYTE;

```

SOURCE CODE FOR NETX25.SRC (MAIN MODULE). 18 NOV 85

```
END RCV$RNR;

RCV$RR:  PROCEDURE (CHANNEL, P$BIT, SEQ$NUM, ADDR) EXTERNAL;
DECLARE (CHANNEL, P$BIT, SEQ$NUM, ADDR) BYTE;
END RCV$RR;

RCV$DISC:  PROCEDURE (CHANNEL, P$BIT) EXTERNAL;
DECLARE (CHANNEL, P$BIT) BYTE;
END RCV$DISC;

RCV$CMDR:  PROCEDURE (CHANNEL, P$BIT) EXTERNAL;
DECLARE (CHANNEL, P$BIT) BYTE;
END RCV$CMDR;

RCV$DM:  PROCEDURE (CHANNEL, P$BIT) EXTERNAL;
DECLARE (CHANNEL, P$BIT) BYTE;
END RCV$DM;

FRAME$PRESENT:  PROCEDURE (CHANNEL) BYTE EXTERNAL;
DECLARE CHANNEL BYTE;
END FRAME$PRESENT;

PRINTI:  PROCEDURE (NUM) EXTERNAL;
DECLARE NUM INTEGER;
END PRINTI;

DSPLY$FRAME$HDR:  PROCEDURE (CHANNEL) EXTERNAL;
DECLARE CHANNEL BYTE;
END DSPLY$FRAME$HDR;

TIMES$DELAY$CHA:  PROCEDURE EXTERNAL;
END TIMES$DELAY$CHA;

TIMES$DELAY$CHB:  PROCEDURE EXTERNAL;
END TIMES$DELAY$CHB;

ROUTE$PACKET:  PROCEDURE EXTERNAL;
END ROUTE$PACKET;

READ$TAB:  PROCEDURE EXTERNAL;
END READ$TAB;

FIND$IFRAME:  PROCEDURE (TABLE) BYTE EXTERNAL;
DECLARE TABLE BYTE;
END FIND$IFRAME;

DECLARE (CHANNEL, P$BIT, SEQ$NUM, CNTL$ERROR, ERROR$STATUS) BYTE;

DECLARE (ACTUAL, STATUS) ADDRESS;
DECLARE STATUS$PTR (81) BYTE;
```

SOURCE CODE FOR NETX25.SRC (MAIN MODULE), 18 NOV 85

```

DECLARE BUFFER (128) BYTE;
DECLARE (R$CONN, W$CONN, F$CONN) ADDRESS PUBLIC;
DECLARE CR LITERALLY 'ODH';
DECLARE LF LITERALLY 'OAH';
DECLARE CR$LF (2) BYTE DATA (ODH, OAH);
DECLARE MESSAGE(*) BYTE DATA(ODH, OAH,
'THIS IS THE TEST MESSAGE
THIS IS THE TEST MESSAGE!!!!');

```

```

/***** END EXTERNALS *****/

```

```

DECLARE FALSE LITERALLY 'OH'; /* USE AS FLAGS TO TEST */
TRUE LITERALLY 'OFFH'; /* BITS FOR BRANCHING */
CONCTC LITERALLY 'ODSH'; /* NETWORK MONITOR CTC PORT ADDRESS */
CONCMD LITERALLY 'ODFH'; /* NETWORK MONITOR USART COMMAND PORT ADDRESS */
CONDAT LITERALLY 'ODEH'; /* NETWORK MONITOR USART DATA PORT ADDRESS */

NET$R1$DST$ERR LITERALLY '10'; /* NET ROUTE$IN DEST ERROR ENTRY */
NET$R0$DST$ERR LITERALLY '11'; /* NET ROUTE$OUT DEST ERROR ENTRY */
PACKET$SIZE LITERALLY '138'; /* PACKET IS 138 BYTE BLOCK */
PACKET$IN$TABLE LITERALLY '10'; /* # PACKETS IN TABLE */
PACKET$TABLE$SIZE LITERALLY '1380';
I$FRAME$SIZE LITERALLY '140';
U$FRAME$SIZE LITERALLY '2';
S$FRAME$SIZE LITERALLY '2';
CMDR$FRAME$SIZE LITERALLY '5';
FRAME$IN$TABLE LITERALLY '10';
FRAME$TABLE$SIZE LITERALLY '1400';
DATA$SIZE LITERALLY '128';
IP$DATA$SIZE LITERALLY '96';
TCP$DATA$SIZE LITERALLY '72';
TEST LITERALLY '0';

```

```

/* NETWORK VARIABLES FOR THIS UNIT */

```

```

/* NOTES: 1. THIS$UNIT$NBR MUST REFLECT WHICH UNIT THIS IS.
2. THIS$COUNTRY$CODE MUST REFLECT THE AREA TO WHICH THIS
UNIT IS LOCATED.
3. MAX$COUNTRY$CODE WILL INDICATE WHICH COUNTRY CODES
ARE CURRENTLY OPERATIONAL. CC = 0000 IS RESERVED
FOR THE DELNET MONITOR.
4. MAX$NETWORK$CODE WILL INDICATE HOW MANY UNITS ARE
CURRENTLY OPERATIONAL WITHIN A PARTICULAR COUNTRY.
5. FOR DETAILED INFORMATION ON THE ABOVE, REFER TO
PHISTER'S THESIS, APPENDIX D.
*/

```

```

DECLARE

```

SOURCE CODE FOR NETX25.SRC (MAIN MODULE), 18 NOV 85

```

THIS$UNID$NBR LITERALLY '02H', /* UNIQUE ADDRESS OF THIS UNID */
THIS$COUNTRY$CODE LITERALLY '01H', /* COUNTRY WHERE THIS UNID RESIDES */
MAX$COUNTRY$CODE LITERALLY '01H', /* COUNTRIES CURRENTLY OPERATIONAL */
MAX$NETWORK$CODE LITERALLY '03H', /* NUMBER OF UNIDS OPERATIONAL IN CC */

STAT$NBR LITERALLY '20', /* NUMBER OF ENTRIES IN STATUS TABLE */

/* VARIABLES USED IN N.MAIN$U2 AND N.IN$IO$U2 */
CTCNOA
  BYTE PUBLIC,
  /* PROGRESSIVE NUMBER OF TIME COUNTS FOR NETWORK CHANNEL A */
CTCNOB
  BYTE PUBLIC,
  /* PROGRESSIVE NUMBER OF TIME COUNTS FOR NETWORK CHANNEL B */
MAXNOA
  LITERALLY '2', /* PREVIOUSLY 64H = 1000 */
  /* MAXIMUM NUMBER OF TIMING COUNTS FOR NETWORK CHANNEL A */
MAXNOB
  LITERALLY '2', /* PREVIOUSLY 64H = 1000 */
  /* MAXIMUM NUMBER OF TIMING COUNTS FOR NETWORK CHANNEL B */
RETRAN$A
  BYTE,
  /* PROGRESSIVE NUMBER OF RETRANSMISSIONS OF A FRAME */
RETRAN$B
  BYTE,
  /* PROGRESSIVE NUMBER OF RETRANSMISSIONS OF A FRAME */
MAXRETRAN$A
  LITERALLY '6', /* PROGRESSIVE NUMBER OF RETRANSMISSIONS OF A FRAME */
MAXRETRAN$B
  LITERALLY '6', /* MAXIMUM NUMBER OF RETRANSMISSIONS OF A FRAME */

DECLARE BUSY$STATUS LITERALLY 'OFFH',
NMBR$MSK LITERALLY '07H',
ESC LITERALLY '1BH',
EOT LITERALLY '04H',
DECLARE ASCII(*) BYTE PUBLIC DATA('0123456789ABCDEF');
DECLARE TEMP(24) WORD PUBLIC INITIAL('*****');
DECLARE MSGNUM BYTE PUBLIC;
DECLARE OUT$TAB$FULL BYTE PUBLIC;
DECLARE A$ADD LITERALLY '0000$0001B', /* LAP B ADDRESS BYTE FORMATS */
B$ADD LITERALLY '0000$0001B';
DECLARE FILE$OUT LITERALLY 'OFFH';
DECLARE READY LITERALLY '1';
DECLARE DONE LITERALLY '2';

DECLARE I$CNTL LITERALLY '0000$0001B', /* LAP B FRAME CONTROL BYTE FORMATS */
RR$CNTL LITERALLY '0000$0001B',
RNR$CNTL LITERALLY '0000$0101B',
REJ$CNTL LITERALLY '0000$1001B',
DM$CNTL LITERALLY '0000$1111B',
SABM$CNTL LITERALLY '0010$1111B',
DISC$CNTL LITERALLY '0100$0011B',
UA$CNTL LITERALLY '0110$0011B',
CMDS$CNTL LITERALLY '1000$0111B';
DECLARE P$BIT$MASK LITERALLY '010H';

```


SOURCE CODE FOR NETX25.SRC (MAIN MODULE), 18 NOV 85

```

TX02NS INTEGER PUBLIC.
TX02NE INTEGER PUBLIC.
TX02SZ INTEGER PUBLIC.

LCNTTB(PACKET$TABLE$SIZE) BYTE PUBLIC.
LCNTNS INTEGER PUBLIC.
LCNTNE INTEGER PUBLIC.
LCNTSZ INTEGER PUBLIC.

NLCTTB(PACKET$TABLE$SIZE) BYTE PUBLIC.
NLCTNS INTEGER PUBLIC.
NLCTNE INTEGER PUBLIC.
NLCTSZ INTEGER PUBLIC.

ISFRAME$QUE(FRAME$TABLE$SIZE) BYTE PUBLIC.
ISFRAME$QUE$NS INTEGER PUBLIC.
ISFRAME$QUE$NE INTEGER PUBLIC.
ISFRAME$QUE$SZ INTEGER PUBLIC.

STATTB(STAT$NBR) BYTE;

/*****
/* SEMAPHORE AND TABLE POINTER DECLARATIONS */
*****/

DECLARE SYSSMEM$BASE LITERALLY '08001H'; /* SYSTEM MEMORY BASE ADDRESS */

DECLARE SYSS$BASE BYTE;

/* DECLARATIONS FOR FLAGS AND TABLES IN SYSTEM MEMORY */

DECLARE
(LSEM$1, LSEM$2, LSEM$3, /* LOCAL TO NET SEMAPHORE AND */
 LSEM$4, NSEM$1, NSEM$2) BYTE PUBLIC. /* NET TO LOCAL SEMAPHORE */

(LPTR$1, LSPARE$1, LPTR$2, LSPARE$2, /* LOC TO NET PACKET PTR */
 LPTR$3, LSPARE$3, LPTR$4, LSPARE$4, /* LOC TO NET PACKET PTR */
 NPTR$1, NSPARE$1, NPTR$2, NSPARE$2) POINTER PUBLIC; /* NET TO LOC PACKET PTR */

/*
DECLARE
FRAME STRUCTURE(ADDRS BYTE,
CONTROL BYTE,
PACKET$DATA(PACKET$SIZE) BYTE);

PACKET STRUCTURE(GFI$LGCN BYTE,
LCN BYTE,
SEQUENCE BYTE,
SRC$DST$LEN BYTE,

```

SOURCE CODE FOR NETX25.SRC (MAIN MODULE). 18 NOV 85

```

SOURCE      BYTE,
DEST        BYTE,
PADDING     BYTE,
FAC$LEN     BYTE,
FAC$CODE    BYTE,
FAC$PARAM   BYTE,
DATAM(DATA$SIZE) BYTE);

```

```

DECLARE      IPDATA  STRUCTURE(VER$IHL  BYTE,
                                TYPE$SERV BYTE,
                                TOT$LEN1  BYTE,
                                TOT$LEN2  BYTE,
                                ID$1      BYTE,
                                ID$2      BYTE,
                                FLAG$FRAG  BYTE,
                                FRAGOFF    BYTE,
                                TIME$LIVE  BYTE,
                                PROTO      BYTE,
                                CHKSUM$1   BYTE,
                                CHKSUM$2   BYTE,
                                SORC$ADDR(4) BYTE,
                                DEST$ADDR(4) BYTE,
                                OPTIONS(3) BYTE,
                                PADDING     BYTE,
                                DATAM(IP$DATA$SIZE) BYTE);

```

```

DECLARE      TCPDATA STRUCTURE(SOURCE$PORT(2)  BYTE,
                                DESTIN$PORT(2)  BYTE,
                                SEQUEN$NUM(4)   BYTE,
                                ACK$NUM(4)      BYTE,
                                DATA$OFF$PLUS  BYTE,
                                RESERV$PLUS      BYTE,
                                WINDOW(2)       BYTE,
                                CHECKSUM(2)      BYTE,
                                URGENT$PTR(2)    BYTE,
                                OPTIONS(3)       BYTE,
                                PADDING          BYTE,
                                DATAM(TCP$DATA$SIZE) BYTE);

```

*/

/* MISCELLANEOUS DECLARATIONS */

```

DECLARE      FOREVER  BYTE PUBLIC,
              DESTINATION WORD PUBLIC,
              SEQ$NUM$A  BYTE PUBLIC,
              SEQ$NUM$B  BYTE PUBLIC,
              SEND$STATE$A  BYTE PUBLIC,
              SEND$STATE$B  BYTE PUBLIC,
              RCV$STATE$A  BYTE PUBLIC,
              RCV$STATE$B  BYTE PUBLIC,

              /* DESTINATION OF A FRAME */
              /* ACKNOWLEDGE VARIABLE DECLARATIONS */

```

SOURCE CODE FOR NETX25.SRC (MAIN MODULE). 18 NOV 85

```

RCV$STATE$B      BYTE PUBLIC;

DECLARE
  STARTUP$HDR(*) BYTE DATA(CR,LF,
    UNID II #3 NETWORK OS',CR,LF,
    VERS 1.2, 18 NOV 1985',CR,LF,
    EXECUTING',CR,LF);

DECLARE
  TP$1(*) BYTE DATA(CR,LF,
    ENTERING INIT$N$TAB, PROCEDURE');
  TP$2(*) BYTE DATA(CR,LF,
    ENTERING INSIO PROCEDURE');
  TP$3(*) BYTE DATA(CR,LF,
    STARTING DO FOREVER LOOP ----->');
  TP$4(*) BYTE DATA(CR,LF,
    ISIDE DO FOREVER LOOP =>');
  TP$5(*) BYTE DATA(CR,LF,
    INCOMING FRAME LOCATED IN NETWORK CHANNEL A');
  TP$6(*) BYTE is located in module LAPB
  TP$7(*) BYTE is located in module LAPB
  TP$8(*) BYTE DATA(CR,LF,
    FRAME IS INCOMING S FRAME');
  TP$9(*) BYTE DATA(CR,LF,
    THIS SEQ$BIT$A = 1');
  TP$10(*) BYTE DATA(CR,LF,
    THIS SEQ$BIT$A = 0');
  TP$11(*) BYTE DATA(CR,LF,
    FRAME IS INCOMING I FRAME');
  TP$12(*) BYTE DATA(CR,LF,
    INPUT SEQ$BIT$A = 1');
  TP$13(*) BYTE DATA(CR,LF,
    INPUT SEQ$BIT$A = 0');
  TP$14(*) BYTE DATA(CR,LF,
    INCOMING FRAME LOCATED IN NETWORK CHANNEL B');
  TP$15(*) BYTE is located in module LAPB
  TP$16(*) BYTE is located in module LAPB
  TP$17(*) BYTE DATA(CR,LF,
    FRAME IS INCOMING S FRAME');
  TP$18(*) BYTE DATA(CR,LF,
    THIS SEQ$BIT$B = 1');
  TP$19(*) BYTE DATA(CR,LF,
    THIS SEQ$BIT$B = 0');
  TP$20(*) BYTE DATA(CR,LF,
    FRAME IS INCOMING I FRAME');
  TP$21(*) BYTE DATA(CR,LF,
    INPUT SEQ$BIT$B = 1');
  TP$22(*) BYTE DATA(CR,LF,
    INPUT SEQ$BIT$B = 0');
  TP$23 TP$24, TP$25 IS IN MODULE PKT
  TP$26(*) BYTE PUBLIC DATA(CR,LF,
    OUTGOING FRAME LOCATED IN NETWORK CHANNEL A');
  TP$26

```



```

DECLARE TP$59(*) BYTE DATA(CR,LF,
      'TP$59
      MSG1(*) BYTE DATA(CR,LF,
      want to load the test message? ');
      MSG2(*) BYTE DATA(CR,LF,
      'Do you want to stop the test? ');
      MSG3(*) BYTE DATA(CR,LF,
      'How many packets to load (1-9)? ');
      MSG4(*) BYTE DATA(CR,LF,
      'Send to which UNID (1 or 3)? ');
      MSG5(*) BYTE DATA(CR,LF,
      'Send any U or S frames? ');
      MSG6(*) BYTE DATA(CR,LF,
      DECLARE
      'ENTER: 0 - SND$CMDR, 1 - SND$SRP, 2 - SND$RNR, 3 - SND$REJ',CR,LF,
      4 - SND$UA, 5 - SND$DISC, 6 - SND$DM, 7 - SND$SABM',CR,LF,
      /*****
      /*****
      * DATE: 30 SEP 84
      * VERSION: 1.0
      * NAME: ERR$CHK
      * MODULE NUMBER: 7.3
      * DESCRIPTION: Checks for errors in the status returned by
      ISIS systems calls. IF an error occurs an orderly exit
      from the program is provided and an ISIS error
      is printed.
      * PASSED VARIABLES: None
      * RETURNS: None
      * GLOBAL VARIABLES USED:
      * GLOBAL VARIABLES CHANGED: None
      * MODULES CALLED:
      * CALLING MODULES:
      * AUTHOR: C.T. Childress
      * HISTORY: 1.0 Capt C.T. Childress - original translated PL/M
      * version
      *
      ERR$CHK: PROCEDURE PUBLIC;
      IF STATUS <> 0 THEN
      DO;
      CALL DQ$DECODE$EXCEPTION ( STATUS, @STATUS$PTR, @STATUS);
      CALL DQ$WRITE (W$CONN, @STATUS$PTR(1), STATUS$PTR(0), @STATUS);
      CALL DQ$WRITE (W$CONN, @CRLF, 2, @STATUS);
      CALL DQ$EXIT (0);
      END;
      END ERR$CHK;
      /*****
      /*****
      DATE: 28 AUG 85

```

SOURCE CODE FOR NETX25.SRC (MAIN MODULE), 18 NOV 85

```

* VERSION: 2.0
* NAME: INIT$NSTAB
* MODULE NUMBER: 6.0
* DESCRIPTION: Initializes the variables used for the network
*               tables, sequence counters, and time out counters.
* PASSED VARIABLES: None
* RETURNS: None
* GLOBAL VARIABLES USED: TIMCHA, TIMCHB, CTCNOA, CTCNOB, FOREVER,
* MSGNUM, RETRANS$A, RETRANS$B, NT01NS, NT01NE,
* NT01SZ, NT02NS, NT02NE, NT02SZ, LCNTNS, LCNTNE,
* LCNTSZ, NTLCSN, NTLCSNE, NTLCSZ, OUFRA$CHASNS,
* TX01NE, TX01SZ, USCMD$QUE$A, USCMD$QUE$B
* TX02NS, TX02NE,
* TX02SZ
* GLOBAL VARIABLES CHANGED: All the above
* MODULES CALLED: None
* CALLING MODULES: main
* AUTHOR: Capt Mark Weber
* HISTORY: 2.0 Capt Mark Weber - changed the sequence variables used
*          1.0 Capt C.T. Childress - original translated PL/M
*****
INIT$NSTAB: PROCEDURE;
*****/

DECLARE INDEX BYTE;

/* VARIABLES FOR BOTH NETWORK CHANNELS */
SEND$STATES$A = FALSE; /* SEQ NUM GOING INTO FRAME */
SEND$STATES$B = FALSE; /* SEQ NUM GOING INTO FRAME */
RCV$STATES$A = FALSE; /* CURRENT SEQUENCE NUM */
RCV$STATES$B = FALSE; /* CURRENT SEQUENCE NUM */

TIMCHA = TRUE; /* FLAG FOR CH A WAIT LOOP */
TIMCHB = TRUE; /* FLAG FOR CH B WAIT LOOP */
CTCNOA = 0;
CTCNOB = 0;

FOREVER = TRUE; /* FLAG FOR MAIN LOOP */
MSGNUM = 0;

RETRANS$A = 0;
RETRANS$B = 0;

NT01NS = 0;
NT01NE = 0;
NT01SZ = FRAME$TABLE$SIZE;

NT02NS = 0;
NT02NE = 0;

```

```

NT02SZ = FRAME$TABLE$SIZE;

TX01NS = 0;
TX01NE = 0;
TX01SZ = FRAME$TABLE$SIZE;

TX02NS = 0;
TX02NE = 0;
TX02SZ = FRAME$TABLE$SIZE;

LCNTNS = 0;
LCNTNE = 0;
LCNTSZ = PACKET$TABLE$SIZE;

NTLCNS = 0;
NTLCNE = 0;
NTLCNZ = PACKET$TABLE$SIZE;

LSEM$1 = DONE; /* INITIALIZATION OF THE SEMAPHORE FLAGS */
LSEM$2 = DONE;
LSEM$3 = DONE;
LSEM$4 = DONE;

NSEM$1 = DONE;
NSEM$2 = DONE;

LSPARE$1 = @LCNTTB(0); /* THE SPARE ADDRESS LOCATIONS SHOULD BE SET TO THE */
LSPARE$2 = @LCNTTB(0); /* SEGMENT NUMBER INTO WHICH THE LOCAL BOARD MEMORY */
LSPARE$3 = @LCNTTB(0); /* WILL BE MAPPED INTO SYSTEM MEMORY. THIS VALUE IS */
LSPARE$4 = @LCNTTB(0); /* CURRENTLY 0 FOR THE LSPARE$X VARIABLES. */

NSPARE$1 = @LCNTTB(0); /* SAME AS FOR LSPARE$X DISCUSSION ABOVE */
NSPARE$2 = @LCNTTB(0);

LPTR$1 = @LCNTTB(0);
LPTR$2 = @LCNTTB(0);
LPTR$3 = @LCNTTB(0);
LPTR$4 = @LCNTTB(0);

NPTR$1 = @LCNTTB(0);
NPTR$2 = @LCNTTB(0);

/* INITIALIZE THE US$CMD$SQUE */
US$CMD$SQUE$A.ACK, US$CMD$SQUE$B.ACK = TRUE;
US$CMD$SQUE$A.CMD, US$CMD$SQUE$B.CMD = DISC$CNTL;

/* INITIALIZE PACKET DEFAULTS USED BY THIS SIMULATION

```

—

```

*****
* DATE: 30 SEP 84 *
* VERSION: 1.0 *
* NAME: ASC$HEX *
* MODULE NUMBER: 7.4 *
* DESCRIPTION: Converts an ASCII representation of a HEX *
*****

```


SOURCE CODE FOR NETX25.SRC (MAIN MODULE), 18 NOV 85

```

*      number into a HEX word.
* PASSED VARIABLES:
*      C - the ASCII
* RETURNS:      ASCII$HEX the HEX work
* GLOBAL VARIABLES USED:
*      None
* GLOBAL VARIABLES CHANGED:
*      None
* MODULES CALLED:
*      None
* CALLING MODULES:
*      READ$LINE
*      Capt C. T. Childress
* HISTORY: 1.0 Capt C.T. Childress - original translated PL/M
*      version
* *****/

```

```

ASCII$HEX: PROCEDURE(C) WORD;
DECLARE C BYTE;

```

```

IF C <= '9' THEN
    RETURN DOUBLE(C-30H);
ELSE
    RETURN DOUBLE(C-37H);

```

```

END ASCII$HEX;
/*****
* DATE:      30 SEP 84
* VERSION:   1.0
* NAME:      VALID$HEX
* MODULE NUMBER:
*      None as yet
* DESCRIPTION:
*      Checks to see if the ASCII characters
*      make up a valid HEX number.
*      H - the ASCII characters
*      VALID$HEX - a flag
* PASSED VARIABLES:
* RETURNS:
*      None
* GLOBAL VARIABLES USED:
*      None
* GLOBAL VARIABLES CHANGED:
*      None
* MODULES CALLED:
*      None
* CALLING MODULES:
*      None as yet
*      Capt C.T. Childress
* HISTORY: 1.0 Capt C.T. Childress - original translated PL/M
*      version
* *****/
VALID$HEX: PROCEDURE (H) BYTE;

```

```

DECLARE (H, I) BYTE;

```

```

DO I = 0 TO LAST(ASCII);
    IF H = ASCII(I) THEN
        RETURN TRUE;

```

```

END;
RETURN FALSE;

```

```

END VALID$HEX;

```

SOURCE CODE FOR NETX25.SRC (MAIN MODULE), 18 NOV 85

```

/*****
* DATE: 30 SEP 84
* VERSION: 1.0
* NAME: SENDSEQ
* MODULE NUMBER: 5.0
* DESCRIPTION: This procedure takes a message string
and outputs it either to the system ISIS console or
when implemented in the SBC 88/45 board to the
SBC 86/12A monitor. The code for the SBC 86/12 A is
currently commented out. The variable FILE$OUT when
set, allows all output to the console to be placed in
the file FILE.OUT for a hard copy.
* PASSED VARIABLES: MSG, TOTAL
* RETURNS: None
* GLOBAL VARIABLES USED: BUFFER (as used by ISIS calls)
* GLOBAL VARIABLES CHANGED: BUFFER (as changed by ISIS calls)
* MODULES CALLED: DQ$WRITE, DQ$READ, ERR$CHK
* CALLING MODULES: main, READLINE, LOAD, ROUTE$IN, ROUTE$OUT
* BUILD$I$FRAME, BUILD$$FRAME, SERVICE$LOOP
* AUTHOR: Capt C.T. Childress
* HISTORY: 1.0 Capt C.T. Childress - original translated PL/M
*
*****/
SENDDSEQ: PROCEDURE (MSG, TOTAL) PUBLIC REENTRANT;
DECLARE MSG POINTER;
DECLARE TOTAL WORD;

CALL DQ$WRITE (W$CONN, MSG, TOTAL, @STATUS);
CALL ERR$CHK;
IF FILE$OUT THEN DO;
CALL DQ$WRITE (F$CONN, MSG, TOTAL, @STATUS);
CALL ERR$CHK;
END;
END SENDDSEQ;

/*****
* DATE: 28 AUG 85
* VERSION: 1.1
* NAME: LD$TAB$H$SKP
* MODULE NUMBER: 7.8.2
* DESCRIPTION: This procedure house keeps the specified
table (TABLE). The procedure determines which table
to process and then advances the table pointer by the
distance FRAME$SIZE. Currently, FRAME$SIZE is always
1$FRAME$SIZE (140 bytes) or PACKET$SIZE so as to keep
a constant number of frames or packets in each table.
This feature will only be needed when variable length
I frames are supported.
I frames in each table.
*
*****/

```

SOURCE CODE FOR NETX25.SRC (MAIN MODULE), 18 NOV 85

```

* PASSED VARIABLES:      TABLE, FRAME$SIZE
* RETURNS:
* GLOBAL VARIABLES USED:  NTO1NE, NTO2NE, NTO1SZ, NTO2SZ,
*                          TXO1NE, TXO1NE,
*                          TXO1SZ, TXO2SZ, NTLONE,
*                          LCNTNE, LCNTSZ
* GLOBAL VARIABLES CHANGED: NTO1NE, NTO2NE, TXO1NE,
*                          TXO2NE, NTLONE, LCNTNE
* MODULES CALLED:         None
* CALLING MODULES:        LOAD, BUILD$I$FRAME, BUILD$S$FRAME, ROUTE$IN,
*                          SERVICE$LOOP
* AUTHOR:
* HISTORY: 1.0 Capt C.T. Childress - original translated PL/M
*
***** version *****
LD$TAB$H$K$P: PROCEDURE(TABLE, FRAME$SIZE) PUBLIC REENTRANT;
*****
DECLARE TABLE BYTE;
DECLARE FRAME$SIZE INTEGER;
DECLARE TP$LD$TAB(*) BYTE DATA(CR,LF,
'TP$LD$TAB
ENTERING LD$TAB$H$K$P
');
IF TEST THEN
DO;
CALL SENDSEQ(@TP$LD$TAB, LENGTH(TP$LD$TAB));
CALL PRINTI(INT(TABLE));
END;
IF (TABLE >= 1 AND TABLE <= 6) THEN
DO CASE TABLE;
: /* ZERO CASE IS NULL AND IS AN ERROR CASE */
DO; /* HSKP INCOMING NET CH A TABLE */
IF NTO1NE = NTO1NE + FRAME$SIZE;
IF NTO1NE >= NTO1SZ THEN /* IF TABLE WRAP */
NTO1NE = 0;
IF TEST THEN CALL PRINTI(NTO1NE);
END;
DO; /* HSKP INCOMING NET CHB TABLE */
NTO2NE = NTO2NE + FRAME$SIZE;
IF NTO2NE >= NTO2SZ THEN /* IF TABLE WRAP */
NTO2NE = 0;
IF TEST THEN CALL PRINTI(NTO2NE);
END;
DO; /* HSKP OUTGOING NET CH A TABLE */
TXO1NE = TXO1NE + FRAME$SIZE;
IF TXO1NE >= TXO1SZ THEN /* IF TABLE WRAP */
TXO1NE = 0;

```

SOURCE CODE FOR NETX25.SRC (MAIN MODULE), 18 NOV 85

```

IF TEST THEN CALL PRINTI(TX01NE);
END;

DO;
    /* HSKP OUTGOING NET CH B TABLE */
    TX02NE = TX02NE + FRAME$SIZE;
    IF TX02NE >= TX02SZ THEN /* IF TABLE WRAP */
        TX02NE = 0;
    IF TEST THEN CALL PRINTI(TX02NE);
END;

DO;
    /* HSKP NETWORK TO LOCAL TABLE */
    NTLNE = NTLNE + PACKET$SIZE;
    IF NTLNE >= NTLCSZ THEN /* IF TABLE WRAP */
        NTLNE = 0;
    IF TEST THEN CALL PRINTI(NTLNE);
END;

DO;
    /* HSKP LOCAL TO NETWORK TABLE */
    LCNTNE = LCNTNE + PACKET$SIZE;
    IF LCNTNE >= LCNTSZ THEN /* IF TABLE WRAP */
        LCNTNE = 0;
    IF TEST THEN CALL PRINTI(LCNTNE);
END;

END;
ELSE HSKP$ERR = HSKP$ERR + 1;
END LD$TAB$HSKP;

/*****
* DATE: 28 AUG 85
* VERSION: 1.1
* NAME: SRVCS$TAB$HSKP
* MODULE NUMBER: 8.3.2
* DESCRIPTION: The purpose of this procedure is to
* housekeep a specified table after servicing a packet or
* frame from a specific table. The variable FRAME$SIZE
* is currently a constant I$FRAME$SIZE or PACKET$SIZE to
* keep the number of frames or packets in a table constant.
* This variable will only be necessary when variable length
* I frames are supported.
* PASSED VARIABLES: TABLE, FRAME$SIZE
* RETURNS: None
* GLOBAL VARIABLES USED: HSKP$ERR, NT01NS, NT02NS, NTLNS,
* LCNTNS, TX01NS, TX02NS
* GLOBAL VARIABLES CHANGED: All above
* MODULES CALLED: PRINTI
* CALLING MODULES: LOAD, ROUTE$IN, ROUTE$OUT,
* AUTHOR: Mark Weber
* HISTORY: 1.1 Mark Weber - 28 AUG 85 - added variable FRAME$SIZE
* 1.0 C.T. Childress - 30 SEP 84 - original translated PL/M*
*****/

```

```

SOURCE CODE FOR NETX25.SRC (MAIN MODULE), 18 NOV 85

*
* version
*
SRVC$TAB$H$SKP: PROCEDURE(TABLE, FRAME$SIZE) PUBLIC REENTRANT;
*****

DECLARE TABLE BYTE;
DECLARE FRAME$SIZE INTEGER;
DECLARE TP$SRVC$TAB(*) BYTE DATA(CR,LF,
'TP$SRVC$TAB
ENTERING SRVC$TAB$H$SKP ');

IF TEST THEN
DO:
CALL SENDSEQ(@TP$SRVC$TAB, LENGTH(TP$SRVC$TAB));
CALL PRINTI(INT(TABLE));
END;
IF TABLE >= 1 AND TABLE <= 6 THEN
DO CASE TABLE:
; /* ZEROth ENTRY IS NULL AND AN ERROR */
DO: /* IF CALLED TO HSKP INCOMING NET CH A TABLE */
NT01NS = NT01NS + FRAME$SIZE;
IF NT01NS >= NT01SZ THEN /* IF TABLE WRAP */
NT01NS = 0;
IF TEST THEN CALL PRINTI(NT01NS);
END;
DO: /* IF CALLED TO HSKP INCOMING NET CH B TABLE */
NT02NS = NT02NS + FRAME$SIZE;
IF NT02NS >= NT02SZ THEN /* IF TABLE WRAP */
NT02NS = 0;
IF TEST THEN CALL PRINTI(NT02NS);
END;
DO: /* IF CALLED TO HSKP OUTGOING NET CH A TABLE */
TX01NS = TX01NS + FRAME$SIZE;
IF TX01NS >= TX01SZ THEN /* IF TABLE WRAP */
TX01NS = 0;
IF TEST THEN CALL PRINTI(TX01NS);
END;
DO: /* IF CALLED TO HSKP OUTGOING NET CH B TABLE */
TX02NS = TX02NS + FRAME$SIZE;
IF TX02NS >= TX02SZ THEN /* IF TABLE WRAP */
TX02NS = 0;
IF TEST THEN CALL PRINTI(TX02NS);
END;
DO: /* IF CALLED TO HSKP NETWORK TO LOCAL TABLE */
NTLCNS = NTLCNS + FRAME$SIZE;
IF NTLCNS >= NTLC$SZ THEN

```

SOURCE CODE FOR NETX25.SRC (MAIN MODULE), 18 NOV 85

```

      NTLCONS = 0;
      IF TEST THEN CALL PRINTI(NTLCONS);
    END;

    DO; /* IF CALLED TO HOUSEKEEP LOCAL TO NET TABLE */
      LCNTNS = LCNTNS + FRAME$SIZE; /* IF TABLE WRAP */
      IF LCNTNS >= LCNTSZ THEN
        LCNTNS = 0;
      IF TEST THEN CALL PRINTI(LCNTNS);
    END;

  END;
ELSE HSKPERR = HSKPERR + 1;
END SRV$TAB$HSKP;
/*****
* DATE: 28 AUG 85
* VERSION: 1.1
* NAME: BUILD$I$FRAME
* MODULE NUMBER: 9.1.11.1
* DESCRIPTION: The purpose of this procedure is to
*              build an I FRAME for a packet received from one of the
*              local hosts. The first two bytes of the frame and
*              network headers are filled out, then the I frame is
*              placed in the appropriate network transmission table.
*              Finally, a packet from LCNTTB is loaded the chosen
*              transmit table. This procedure simulates the process
*              needed to move a packet from one of the four SBC 544
*              channels to the SBC 88/45 transmit channel.
*
* PASSED VARIABLES:
* RETURNS: None
* GLOBAL VARIABLES USED: HSKPERR, NT01NS, NT02NS, NTLCONS,
*                        LCNTNS, TX01NS, TX02NS
*
* GLOBAL VARIABLES CHANGED: All above
* MODULES CALLED: None
* CALLING MODULES: ROUTES$IN
* AUTHOR: Mark Weber
* HISTORY: 1.1 Mark Weber - 28 AUG 85 - changed header format
*          1.0 C.T. Childress - 30 SEP 84 - original translated PL/M
*          version
*****
BUILD$I$FRAME: PROCEDURE(TABLE) PUBLIC REENTRANT;

  DECLARE TABLE WORD,
  DEST$BYTE BYTE;

  IF TABLE >= 1 AND TABLE <= 2 THEN
    DO CASE TABLE;

      ; /* ZEROth ENTRY IS NULL AND AN ERROR */

```

SOURCE CODE FOR NETX25.SRC (MAIN MODULE), 18 NOV 85

```
DO: /* I FRAME IS GOING OUT NETWORK CHANNEL A */
CALL SENDSEQ(@TP$38, LENGTH(TP$38));
NT01TX (TX01NE+ 0) = BSADD;
/* Take the receive state variable and place it in the uppermost
three bits of the control byte. Take the send state
variable and place it in bits 4 through 2 ( total is 8 to
1 with 8 being MSB )
NT01TX (TX01NE+ 1) = (((SHL(RCV$STATE$A, 5) AND 0E0H) OR 010H) OR
(SHL(SEND$STATE$A, 1) AND 0EH));
CALL MOV(B @LCNTTB (LCNTNS), @NT01TX (TX01NE+2),
PACKET$SIZE);
CALL LD$TAB$H$SKP (3, IFRAME$SIZE);
END;
```

```
DO: /* I FRAME IS GOING OUT NETWORK CHANNEL B */
CALL SENDSEQ(@TP$39, LENGTH(TP$39));
NT02TX (TX02NE+ 0) = ASADD;
/* Take the receive state variable and place it in the uppermost
three bits of the control byte. Take the send state
variable and place it in bits 4 through 2 ( total is 8 to
1 with 8 being MSB )
NT02TX (TX02NE+ 1) = (((SHL(RCV$STATE$B, 5) AND 0E0H) OR 010H) OR
(SHL(SEND$STATE$B, 1) AND 0EH));
CALL MOV(B @LCNTTB (LCNTNS), @NT02TX (TX02NE+2),
PACKET$SIZE);
CALL LD$TAB$H$SKP (4, IFRAME$SIZE);
END;
```

END;
END BUILD\$I\$FRAME;

```
/* ***** 31 AUG 85 ***** */
* DATE: 31 AUG 85 *
* VERSION: 1.1 *
* NAME: SERVICES$LOOP *
* MODULE NUMBER: 9.3.2.1 *
* DESCRIPTION: This procedure serves as dummy trans-
* mit routines for the high speed datalink layer soft-
* ware. The routine loops back I frames by exchanging
* source and destination headers. When an I frame is
* found the source and destination address are swapped
* and the I frame is looped to the opposing channel
* receive table (i.e. frames from channel A are sent to
* channel B receive table). The same process is followed
* for S and U frames except for RR frames which are
* discarded. RR frames are not looped back as they are
* tested through the RCV$I$FRAME procedure.
* PASSED VARIABLES: TABLE, FRAME$SIZE
* RETURNS: NONE
* GLOBAL VARIABLES USED: NT01NE, TX01NE, TX02NE, NT02RX
* NT01NS, NT02NS, TX01NS, TX02NS, TX01RX, NT02RX *
```

SOURCE CODE FOR NETX25.SRC (MAIN MODULE), 18 NOV 85

```

* GLOBAL VARIABLES CHANGED:  NT01RX, NT02RX
* MODULES CALLED:  LD$TAB$H$SKP
* CALLING MODULES:
* AUTHOR:  Mark Weber
* ALOGRITHM:  IF frame present in transmit table then
*             copy frame to intermediate storage
*             IF NOT I frame then
*             Loop all S/R frames except RR frames
*             Update receive table
*             ELSE
*             Copy frame to receive table
*             Change network header source and destination
*             address
*             Change TCP/IT header source and destination
*             address
*             Update receive table
*             END IF
*             END IF
* HISTORY:  1.1 Mark Weber - 30 AUB 85- Changed procedure to loop
*           back S and U frames
*           1.0 C.T. Childress - 30 SEP 85- original translated PL/M
*
*****
SERVICE$LOOP: PROCEDURE(TABLE, FRAME$SIZE) REENTRANT;
DECLARE CHANNEL BYTE,
        ADDR(1$FRAME$SIZE) BYTE,
        FRAME$SIZE INTEGER,
        TABLE BYTE,
        INDEX INTEGER,
        ORIGIN INTEGER;

ORIGIN = 0;

DO CASE TABLE;
; /* ZERO IS NULL CASE */
DO:
IF ((TX01NE - TX01NS) >= FRAME$SIZE) OR /* FRAME PRESENT*/
(TX01NS > TX01NE) THEN DO; /*If frame present, loop to channel B*/

DO INDEX = 0 TO (FRAME$SIZE - 1);
ADDR(INDEX) = NT01TX(TX01NS + INDEX);
END;

IF (ADDR(ORIGIN + 1) AND I$CNTL) = 1 THEN /* CHECK FOR S OR U FRAME */
DO:
DO INDEX = 0 TO (FRAME$SIZE - 1); /* DO'NT LOOP RR FRAMES */
NT02RX(NT02NE + INDEX) = ADDR(ORIGIN + INDEX);
END;

```


SOURCE CODE FOR NETX25.SRC (MAIN MODULE), 18 NOV 85

```

CALL LD$TAB$H$SKP(2, I$FRAME$SIZE);
IF TEST THEN CALL SENDSEQ(@TP$53B, LENGTH(TP$53B));
/* LOOP THE S / U FRAME */

END;
ELSE DO;
/* HAVE I FRAME */
CALL SENDSEQ(@TP$53A, LENGTH(TP$53A));
DO INDEX = 0 TO (FRAME$SIZE - 1);
/* SWAP DATA */
NT02RX(NT02NE + INDEX) = ADDR(ORIGIN + INDEX);
END;
NT02RX(NT02NE + 3) = ROL(ADDR(ORIGIN + 3), 4); /* SWAP FRAME HEADER */
NT02RX(NT02NE + 6) = ADDR(7);
NT02RX(NT02NE + 7) = ADDR(6);
DO INDEX = 0 TO 3;
/* SWAP IP DEST & SOURCE */
NT02RX(NT02NE + INDEX + 24) = ADDR(ORIGIN + INDEX + 28);
NT02RX(NT02NE + INDEX + 28) = ADDR(ORIGIN + INDEX + 24);
END;
CALL LD$TAB$H$SKP(2, I$FRAME$SIZE);
END;
/* END ELSE */
/* END IF FRAME PRESENT */
/* CASE = 1 */

END;

/* CASE 2 */
DO;
IF ((TX02NE - TX02NS) >= FRAME$SIZE) OR
(TX02NS > TX02NE) THEN DO; /* If frame present, loop to channel A */
DO INDEX = 0 TO (FRAME$SIZE - 1);
ADDR(INDEX) = NT02TX(TX02NS + INDEX);
END;

IF (ADDR(ORIGIN + 1) AND I$CNTL) = 1 THEN /* CHECK FOR S OR FRAME */
DO;
DO INDEX = 0 TO (FRAME$SIZE - 1); /* LOOP EVERYTHING ELSE */
NT01RX(NT01NE + INDEX) = ADDR(ORIGIN + INDEX);
END;
CALL LD$TAB$H$SKP(1, I$FRAME$SIZE);
IF TEST THEN CALL SENDSEQ(@TP$53B, LENGTH(TP$53B));
/* LOOP THE S / U FRAME */

END;
ELSE DO;
CALL SENDSEQ(@TP$53A, LENGTH(TP$53A));
DO INDEX = 0 TO (FRAME$SIZE - 1);
/* COPY THE DATA */
NT01RX(NT01NE + INDEX) = ADDR(ORIGIN + INDEX);
END;
NT01RX(NT01NE + 3) = ROL(ADDR(ORIGIN + 3), 4); /* SWAP FRAME HEADER */
NT01RX(NT01NE + 6) = ADDR(ORIGIN + 7);
NT01RX(NT01NE + 7) = ADDR(ORIGIN + 6);
DO INDEX = 0 TO 3;
/* SWAP IP DEST & SOURCE */
NT01RX(NT01NE + INDEX + 24) = ADDR(ORIGIN + INDEX + 28);
NT01RX(NT01NE + INDEX + 28) = ADDR(ORIGIN + INDEX + 24);
END;

```

SOURCE CODE FOR NETX25.SRC (MAIN MODULE), 18 NOV 85

```

      CALL LD$TAB$H$SKP( 1, 1$FRAME$SIZE );
    END;
    END;
    /* END IF FRAME PRESENT */
    /* CASE = 2 */
  END;
  /* CASE TABLE */

END SERVICE$LOOP;

/*****
* DATE: 3 SEP 85
* VERSION: 1.0
* NAME: SERVICE$XMIT$A
* MODULE NUMBER: 9.3.2
* DESCRIPTION: This is a dummy module. The fully
  implemented module will contain the transmit interrupts.
  Currently this module calls a loop back procedure which
  sends the transmit message into the receive table.
* PASSED VARIABLES: FRAME$SIZE
* RETURNS: None
* GLOBAL VARIABLES USED: None
* GLOBAL VARIABLES CHANGED: None
* MODULES CALLED: SERVICE$LOOP
* CALLING MODULES: ROUTE$OUT
* AUTHOR:
* HISTORY: 1.0 Capt C.T. Childress - original translated PL/M
  version
*****/
SERVICE$XMIT$A: PROCEDURE (FRAME$SIZE);

  DECLARE FRAME$SIZE INTEGER;

  CALL SERVICE$LOOP(1,FRAME$SIZE);

END SERVICE$XMIT$A;

/*****
* DATE: 3 SEP 85
* VERSION: 1.0
* NAME: SERVICE$XMIT$B
* MODULE NUMBER: 9.3.4
* DESCRIPTION: This is a dummy procedure. The
  fully implemented procedure will have the interrupt
  routines for channel B. Currently, a loop back pro-
  cedure is called to return the message back to the
  receive table.
* PASSED VARIABLES: FRAME$SIZE
* RETURNS: None
* GLOBAL VARIABLES USED: None
* GLOBAL VARIABLES CHANGED: None
*****/

```

SOURCE CODE FOR NETX25.SRC (MAIN MODULE), 18 NOV 85

```

* MODULES CALLED:          SERVICE LOOP
* CALLING MODULES:        ROUTESOUT
* AUTHOR:
* HISTORY: 1.0 Capt C.T. Childress - original translated PL/M
* version
*****
SERVICES$XMIT$B: PROCEDURE(FRAME$SIZE);
*****
DECLARE FRAME$SIZE INTEGER;

CALL SERVICE$LOOP(2,FRAME$SIZE);

END SERVICE$XMIT$B;
*****
* DATE:          3 SEP 85
* VERSION:       1.0
* NAME:          LOAD
* MODULE NUMBER: 7.5
* DESCRIPTION:   This procedure loads the test messages
*               into the local receive channel buffer for routing to the
*               network transmit table. This procedure is used for
*               simulation purposes only. In operational software,
*               tables LCNTTB and NLCTB would be replaced by the
*               transmit and receive tables on the SBC 544.
*
* PASSED VARIABLES: DEST$UNID
* RETURNS:          None
* GLOBAL VARIABLES USED: LCNTNE, UNID$NBR, THISSUNID$NBR, LCNTTB
* GLOBAL VARIABLES CHANGED: LCNTNE, MGSNUM, LCNTTB
* MODULES CALLED:  LD$TAB$H$SKP, HEX$ASC, SENDSEQ
* CALLING MODULES: READ$LINE
* AUTHOR:
* HISTORY: 1.0 Capt C.T. Childress - original translated PL/M
* version
*****
LOAD: PROCEDURE(DEST$UNID);
*****
DECLARE INDEX INTEGER;
DECLARE DEST$UNID WORD;
DECLARE TEMP$L(60) WORD;

DO INDEX = 0 TO (PACKET$SIZE - 1);
    LCNTTB(LCNTNE + INDEX) = '*';
END;

/* PACKET HEADER */
LCNTTB(LCNTNE + 0) = 010H; /* GF1/LGCN */
LCNTTB(LCNTNE + 1) = (LOW(DEST$UNID AND 0FH) OR
                     (ROL(THISSUNID$NBR, 4) AND 0F0H)); /*SRC, DST */
LCNTTB(LCNTNE + 2) = 0; /* SEQUENCE NUMBER */

```

SOURCE CODE FOR NETX25.SRC (MAIN MODULE). 18 NOV 85

```

LCNTTB(LCNTNE + 3) = 044H; /* DST/SRC FIELD LENGTH */
LCNTTB(LCNTNE + 4) = (LOW(DEST$UNID) AND 0FH) OR 10H; /* DEST = 1, CH = 1 */
LCNTTB(LCNTNE + 5) = (THIS$UNID$NBR AND 0FH) OR 10H; /* SRC = 3, CH = 1 */
LCNTTB(LCNTNE + 6) = 0; /* PADDING */
LCNTTB(LCNTNE + 7) = 010H; /* FACILITY FIELD LENGTH */
LCNTTB(LCNTNE + 8) = 0; /* FACILITY CODE */
LCNTTB(LCNTNE + 9) = 0; /* FACILITY PARAMETER */

/* IP SOURCE HEADER */
LCNTTB(LCNTNE + 22) = 10H; /* CT = 0, CC = 1 */
LCNTTB(LCNTNE + 23) = (THIS$UNID$NBR AND 0FH) OR 60H; /* NC = 3, HC(L) = 6 */
LCNTTB(LCNTNE + 24) = 70H; /* HC(H) = 0, PC(0) = 7 */
LCNTTB(LCNTNE + 25) = 00H; /* PC(1) = 0, PC(2) = 0 */

/* IP DESTINATION HEADER */
LCNTTB(LCNTNE + 26) = 10H; /* CT = 0, CC = 1 */
LCNTTB(LCNTNE + 27) = (LOW(DEST$UNID) AND 0FH) OR 60H; /* NC = 1, HC(L) = 6 */
LCNTTB(LCNTNE + 28) = 70H; /* HC(H) = 0, PC(0) = 7 */
LCNTTB(LCNTNE + 29) = 00H; /* PC(1) = 0, PC(2) = 0 */

CALL MOVB(@MESSAGE, @LCNTTB(LCNTNE + 66), TCP$DATA$SIZE);
IF MSGNUM > 9 THEN
    MSGNUM = 0;
    LCNTTB(LCNTNE + 96) = '0' + MSGNUM;
    MSGNUM = MSGNUM + 1;
/* Display the frame to be sent */
DO INDEX = 0 TO 29;
    IF (INDEX <= 9) OR (INDEX >= 22) THEN
        TEMP$L(INDEX) = HEX$ASC(LCNTTB(LCNTNE + INDEX));
    ELSE
        TEMP$L(INDEX) = '***';
END;
CALL SENDSEQ(@{CR,LF},2);
CALL SENDSEQ(@LCNTTB(LCNTNE + 66), TCP$DATA$SIZE);
CALL SENDSEQ(@{CR,LF},2);
CALL SENDSEQ(@TEMP$L, 30);
CALL SENDSEQ(@{CR,LF},2);
CALL LD$TAB$H$SKP(6, PACKET$SIZE);
CALL SENDSEQ(@TP$54, LENGTH(TP$54));

END LOAD;

/*****
* DATE: 30 SEP 84
* VERSION: 1.0
* NAME: INITACK
* MODULE NUMBER: 9.3.6
* DESCRIPTION: This procedure resets the transmit I
* frame flags, counters, and calls SRV$TAB$H$SKP to up-
*
*****/

```

SOURCE CODE FOR NETX25.SRC (MAIN MODULE), 18 NOV 85

```

* the next to service pointers of the specified table.
* PASSED VARIABLES:
* RETURNS:
* GLOBAL VARIABLES USED:
* GLOBAL VARIABLES CHANGED:
* MODULES CALLED:
* CALLING MODULES:
* AUTHOR:
* HISTORY: 1.0 Capt C.T. Childress - original translated PL/M
*
*****
INITACK: PROCEDURE(TABLE) PUBLIC REENTRANT;

```

```

DECLARE TABLE BYTE;

```

```

IF (TABLE >= 3) AND (TABLE <= 4) THEN DO;
  IF FRAME$PRESENT(TABLE - 2) THEN CALL SRVC$TAB$H$SKP(TABLE, I$FRAME$SIZE);
  DO CASE TABLE;

```

```

    : /* ZEROth ENTRY IS NULL AND AN ERROR */
    : /* FIRST ENTRY IS NULL AND AN ERROR */
    : /* SECOND ENTRY IS NULL AND AN ERROR */

```

```

  DO;
    TIMCHA = TRUE;
    CTCNOA = 0;
    RETRANS$A = 0;
  END;

```

```

  DO;
    TIMCHB = TRUE;
    CTCNOB = 0;
    RETRANS$B = 0;
  END;

```

```

  END; /* END CASE */
  END; /* END IF */

```

```

END INITACK;

```

```

/*****
* DATE: 29 AUG 85
* VERSION: 1.0
* NAME: ROUTE$OUT
* MODULE NUMBER: 9.2
* DESCRIPTION: This procedure sends frames out of the
* the transit buffers in the order in which they occur.
*
*****/

```

SOURCE CODE FOR NETX25.SRC (MAIN MODULE). 18 NOV 85

```

*
* The buffers are implemented as circular FIFOs. When a
* a frame is next to transmit, the procedure looks at the
* frame control byte. If the frame is an information
* the SEND$STATES$A and SEND$STATES$B counters are updated.
* If the frame is a S$FRAME, the psuedo time out mechan-
* ism is started. If a U FRAME is sent, a positive
* ACKNOWLEDGEMENT is always returned. The program first
* checks for the presence of a frame in the transmit
* table (module FRAME$PRESENT is true), then the pro-
* cedue checks the first and second bytes of the received
* frame. If the address byte is correct for the received
* channel, the processing continues, otherwise, the
* RETRANS error count is updated. For valid address
* bytes, the procedure goes on to determine the length
* of the frame received. The frame length information is
* then sent to the transmt procedures, which in this
* simulation, merely loop the sent data back to the
* receive tables.
*
* This procedure calls SERVICE$XMIT$A/B to per-
* form dummy transmission processing. Procedure INITACK
* handles the TI timer simulation. TIME$DELAY$CHA/CHA
* gives a simulated delay.
*
* NOTE:
* 1. There are two very important parameters relating
* to this procedure that must be considered prior to the
* modification of this module. First, the variable
* MAX$NETWORK$CODE. All active UNIDs on the network will
* start with 0 and proceed to increment in a clockwise
* manner. In this way, the addresses can be compared
* to MAX$NETWORK$CODE to be sure that Non-existent UNIDs
* will not appear in transmitted frames.
*
* 2. The seconde parameter is MAXNUM. MAXNUM is the
* number of times the time out cycly loops through the
* retransmission period for an I frame. This period
* allows time for a valid acknowledgement to be received
* after the I frame is transmitted. The basic period is
* set through N.SIO$U2 for 27 milliseconds. IF MAXNUM =
* 10, then the wait loop is 270 milliseconds.
*
* PASSED VARIABLES: None
* RETURNS: None
* GLOBAL VARIABLES USED: TX01NE, TX02NE, TX01NS, TX02NS, MAXNUM,
* RETRANS$A/B, MAXRETRANS$A/B
* GLOBAL VARIABLES CHANGED: None
* MODULES CALLED: DSPLY$FRAME$HDR, SERVICE$XMITA/B,
* TIME$DELAY$CHA/CHB
* CALLING MODULES: START$DATA$XFER, START$DM$MODE
* ALOGRITHM: IF FRAME$PRESENT channel a then
* DSPLY$FRAME$HEADER
* IF NOT I frame then
* IF command address or response address then
* IF CMDR frame then FRAME$SIZE = CMDR SIZE
*

```

SOURCE CODE FOR NETX25.SRC (MAIN MODULE), 18 NOV 85

```

*      ELSE FRAME$SIZE = $$FRAME$SIZE
*      SERVICE the channel
*      RESET the timer
*      END IF
*      ELSE I frame
*      IF time out false then update the time delay
*      ELSE SERVICE the channel
*      UPDATE the send state variavle
*      Increment the retransmit variable
*      Update the time delay
*      END IF
*      END IF
*      END IF
*      IF FRAME$PRESENT channel B then
*      SERVICE THE SAME AS CHANNEL A
*      END IF
*      AUTHOR: Mark Weber
*      HISTORY: 1.0 Mark Weber -29 AUG 85-original PL/M version.
*      *****
*      ROUTESOUT: PROCEDURE REENTRANT;
*      DECLARE (FRAME$SIZE, NEXT$FRAME) INTEGER;
*      DECLARE (CNTL$BYTE$, CNTL$BYTE$, CNTL$BYTE$) BYTE;
*      DECLARE TP$27A(*) BYTE DATA(CR,LF,
*      'TP$27A <==CONTROL BYTE IS NOT AN I FRAME==> ');
*      /*****CHANNEL A PROCESSING*****/
DO;
  IF FRAME$PRESENT(1) THEN
    CALL SEND$SEQ(@TP$26, LENGTH(TP$26));
    CALL DSPLY$FRAMES$HDR(3);
    IF (((NT01TX (TX0INS) AND ASADD) = ASADD) OR
        ((NT01TX (TX0INS) AND BSADD) = BSADD)) THEN
      DO;
        CNTL$BYTE$ = NT01TX(TX0INS + 1);
        IF ((CNTL$BYTE$ AND 01H) = 1) THEN
          DO;
            IF (((CNTL$BYTE$ AND CMDR$CNTL) = CMDR$CNTL) OR
                ((CNTL$BYTE$ AND CMDR$CNTL) = (CMDR$CNTL OR 010H))) THEN
              FRAME$SIZE = CMDR$FRAME$SIZE;
            ELSE FRAME$SIZE = US$FRAME$SIZE;
            CALL SERVICES$XMIT$(FRAME$SIZE);
            CALL INITACK(3);
            CALL SEND$SEQ(@TP$27, LENGTH(TP$27));
            CALL SEND$SEQ(@TP$27A, LENGTH(TP$27A));
          END;
        ELSE DO;
          IF TIMCHA = FALSE THEN
            CALL TIME$DELAY$CHA;
          ELSE
            /* ITS AN I FRAME */
          /* ITS AN I FRAME */
        END;
      END;
    END;
  END;

```

```

DO; CALL SENDSEQ(@TP$27, LENGTH(TP$27));
IF NOT RNR$MODE$A THEN
  CALL SERVICE$XMIT$(I$FRAME$SIZE); /* SERVICE THE */
  RETRAN$A = RETRAN$A + 1; /* TIME OUT */
  CALL TIME$DELAY$CHA;
END;
IF ((TX0INS + I$FRAME$SIZE) >= FRAME$TABLE$SIZE)
  THEN NEXT$FRAME = 0;
ELSE NEXT$FRAME = TX0INS + I$FRAME$SIZE;
IF (((NT01TX(NEXT$FRAME + 1) AND 1) = 1) AND
  (((TX0INE - NEXT$FRAME) >= I$FRAME$SIZE) OR
  (NEXT$FRAME > TX0INE))) THEN
DO; CALL MOVB(@NT01TX(TX0INS), @I$FRAME$QUE(I$FRAME$QUE$NE),
  I$FRAME$SIZE);
  CALL MOVB(@NT01TX(NEXT$FRAME), @NT01TX(TX0INS),
  I$FRAME$SIZE);
  CALL MOVB(@I$FRAME$QUE(I$FRAME$QUE$NE), @NT01TX(NEXT$FRAME),
  I$FRAME$SIZE);
END;
/* END NTOXTX = 80H */
END; /* END $ADD COMPARISON */
ELSE DO;
  CALL SENDSEQ(@TP$48, LENGTH(TP$48));
  STATTB(11) = STATTB(11) + 1;
  STATTB(14) = STATTB(14) + 1;
  CALL INITACK(3);
  /* END ADDRESS COMPARISON */
  IF RETRAN$A >= MAXRETRAN$A THEN DO;
    CALL INITACK(3);
    CALL SENDSEQ(@TP$56, LENGTH(TP$56));
    CALL SND$CMDR(1,1,CNTL$BYTE$A,1);
  END;
  /* END (TX0INE - TX0INS) CONDITION */
END; /* SERVICE CHAN B */

IF FRAME$PRESENT(2) THEN
DO; CALL SENDSEQ(@TP$28, LENGTH(TP$28));
  CALL D$PLY$FRAME$HDR(4);
  IF (((NT02TX(TX02NS) AND B$ADD) = B$ADD) OR
  ((NT02TX(TX02NS) AND A$ADD) = A$ADD)) THEN
  DO; CNTL$BYTE$B = NT02TX(TX02NS + 1);
    IF ((CNTL$BYTE$B AND 01H) = 1) THEN
    DO; IF(((CNTL$BYTE$B AND CMDR$CNTL) = CMDR$CNTL) OR
      ((CNTL$BYTE$B AND CMDR$CNTL) = (CMDR$CNTL OR 010H))) THEN

```



```

FRAME$SIZE = CMDR$FRAME$SIZE;
ELSE FRAME$SIZE = US$FRAME$SIZE;
CALL SERVICE$XMIT$(FRAME$SIZE);
CALL INITACK(4);
CALL SENDSEQ(@TP$29, LENGTH(TP$29));
CALL SENDSEQ(@TP$27A, LENGTH(TP$27A));
END;
ELSE DO;
  IF TIMCHB = FALSE THEN
    CALL TIME$DELAY$CHB;
  ELSE
    DO;
      CALL SENDSEQ(@TP$29, LENGTH(TP$29));
      IF NOT RNR$MODE$B THEN
        CALL SERVICE$XMIT$(IF$FRAME$SIZE);
      RETRANS$B = RETRANS$B + 1;
      CALL TIME$DELAY$CHB;
    END;
  IF ((TX02NS + IF$FRAME$SIZE) <= FRAME$TABLE$SIZE)
    THEN NEXT$FRAME = 0;
  ELSE NEXT$FRAME = TX02NS + IF$FRAME$SIZE;
  IF (((NT02TX(NEXT$FRAME + 1) AND 1) = 1) AND
    (((TX02NE - NEXT$FRAME) >= IF$FRAME$SIZE) OR
    (NEXT$FRAME > TX02NE))) THEN
    DO;
      CALL MOV$(@NT02TX(TX02NS), @IF$FRAME$QUE(IF$FRAME$QUE$NE),
        IF$FRAME$SIZE);
      CALL MOV$(@NT02TX(NEXT$FRAME), @NT02TX(TX02NS),
        IF$FRAME$SIZE);
      CALL MOV$(@IF$FRAME$QUE(IF$FRAME$QUE$NE), @NT02TX(NEXT$FRAME),
        IF$FRAME$SIZE);
    END;
  END; /* END NTOXTX = 80H */
END; /* END ADDRESS COMPARISON */
ELSE DO;
  CALL SENDSEQ(@TP$47, LENGTH(TP$47));
  STATTB(11) = STATTB(11) + 1;
  STATTB(14) = STATTB(14) + 1;
  CALL INITACK(4);
  /* END B$ADD COMPARISON */
  IF RETRANS$B >= MAXRETRANS$B THEN DO;
    CALL INITACK(4);
    CALL SENDSEQ(@TP$57, LENGTH(TP$57));
    CALL SDC$CMR(2,1,CNTL$BYTES$B,1);
  END;
END; /* END (TX02NE - TX02NS) CONDITION */

END ROUTE$OUT;

/*****

```

SOURCE CODE FOR NETX25.SRC (MAIN MODULE), 18 NOV 85

```

*
* DATE: 28 AUG 85
* VERSION: 1.0
* NAME: ROUTESIN
* MODULE NUMBER: 0.1
* DESCRIPTION: This procedure decodes the received
* control byte and processes each frame in accordance to
* the type of frame received. The procedure first
* examines the local host receive tables for packets
* destined for the UNID datalink layer. A packet is
* then examined for the shortest route to the destina-
* tion. The appropriate channel is then loaded with
* the packet. Each datalink layer receive table is then
* examined for a frame present in the table. If a frame
* is found, the control byte is decoded and the appro-
* priate receive routine is then invoked. When the
* ROUTESIN procedure is invoked while not in the SABM
* mode, only a SABM command is allowed. Any other
* received frames cause the DM response to be sent.
* Should an invalid format control byte be encountered,
* an error message is printed and a CMDF frame is gener-
* ated.
*
* PASSED VARIABLES: CONTROL$BYTE$CHA, CONTROL$BYTE$CHB,
* P$BIT$A, P$BIT$B,
* RETURNS: None
* GLOBAL VARIABLES USED: SABM$MODE$A, SABM$MODE$B, DESTINATION
* OUT$TAB$FULL, NT01NE, NT02NE, NT02NS, NT02NS
* GLOBAL VARIABLES CHANGED: DESTINATION, OUT$TAB$FULL
* MODULES CALLED: REC$I$FRAME, RCV$RR, RCV$REJ, RCV$RNR,
* RCV$UA, RCV$DISC, RCV$CMDF, RCV$SABM, RCV$DM, SND$DM,
* SND$CMDF, BUILD$I$FRAME, SRVC$TAB$H$SKP, FIND$I$FRAME
* START$INFO$XFER
* CALLING MODULES:
* ALGORITHM:
* IF frame in channel A transmit table then
* IF in SABM$MODE$A then
* IF RCV$SABM frame then enter SABM mode
* ELSE send DM response
* END IF
* ELSE IF I frame then process I frame
* ELSE IF UA frame then process UA frame
* ELSE IF DISC frame then process DISC frame
* ELSE IF CMDF frame then process CMDF frame
* ELSE IF SABM frame then process SABM frame
* ELSE IF RNR frame then process RNR frame
* ELSE IF REJ frame then process REJ frame
* ELSE IF RR frame then process RR frame
* ELSE send CMDF response and send error message
* END IF
* IF frame in channel B transmit table then
* PROCESS THE SAME AS WITH CHANNEL A
* USING THE CHANNEL B PROCEDURES

```

SOURCE CODE FOR NETX25.SRC (MAIN MODULE). 18 NOV 85

```

*      Call ROUTESPACET to network tables
*
*  AUTHOR:  Mark Weber
*  HISTORY:  1.1 Mark Weber -28 AUG 85-added LAP 8 procedures
*           1.0 C.T. Chidress -30 SEP 84- original translated PL/M
*
* ***** version *****
ROUTESIN: PROCEDURE REENTRANT;
*****
DECLARE TP$64(*) BYTE DATA(CR,LF,
'TP$64 Entered ROUTESIN Module');
DECLARE TP$65(*) BYTE DATA(CR,LF,
'TP$65 INVALID CONTROL BYTE FOUND IN FRAME ON CHA');
DECLARE TP$66(*) BYTE DATA(CR,LF,
'TP$66 INVALID ADDRESS BYTE FOUND IN FRAME ON CHA');
DECLARE TP$67(*) BYTE DATA(CR,LF,
'TP$67 INVALID CONTROL BYTE FOUND IN FRAME ON CHB');
DECLARE TP$68(*) BYTE DATA(CR,LF,
'TP$68 UNVALID ADDRESS BYTE FOUND IN FRAME ON CHB');
DECLARE TP$69(*) BYTE DATA(CR,LF,
'TP$69 INSIDE CHANNEL A OF ROUTE IN');
DECLARE TP$70(*) BYTE DATA(CR,LF,
'TP$70 INSIDE CHANNEL B OF ROUTE IN');
DECLARE (CONTROL$BYTES$CHA, CONTROL$BYTES$CHB) BYTE;
DECLARE (P$BIT$A, P$BIT$B)
DECLARE (SEQ$NUM$A, SEQ$NUM$B)
DECLARE (ADDRESS$A, ADDRESS$B)

*****CHANNEL A PROCESSING*****
CALL SENDSEQ(@('CR,LF'), 2);
CALL SENDSEQ(@('ROUTESIN CHA NS & NE ='), 22);
CALL PRINTI(NT01NS);
CALL PRINTI(NT01NE);
CALL SENDSEQ(@('CR,LF'), 2);

ADDRESS$A = NT01RX(NT01NS);

IF(((NT01NE - NT01NS) >= I$FRAME$SIZE) OR (NT01NS > NT01NE)) THEN
  IF (ADDRESS$A = A$ADD) OR (ADDRESS$A = B$ADD) THEN
    DO;
    /*****FRAME IS PRESENT IN RECEIVE TABLE*****/
    CALL SENDSEQ(@TP$66B, LENGTH(TP$66B));
    CONTROL$BYTES$CHA = NT01RX(NT01NS + 1);
    SEQ$NUM$A = SHR((CONTROL$BYTES$CHA AND 0E0H), 5);
    P$BIT$A = SHR((CONTROL$BYTES$CHA AND 010H), 4);
    CALL D$PLV$FRAME$HDR(1);
    IF NOT SABM$MODE$A THEN
      DO;
      IF ((CONTROL$BYTES$CHA AND SABM$CNTL) = SABM$CNTL)
        THEN CALL RCV$SABM(1,1);

```

SOURCE CODE FOR NETX25.SRC (MAIN MODULE), 18 NOV 85

```

      ELSE CALL SNOSDM(1,1);

END;
ELSE IF ((CONTROL$BYTESCHA AND 01H) = 0) THEN /*I FRAME*/
  CALL RCV$FRAME(1,PSBIT$A); /*UA FRAME*/
ELSE IF ((CONTROL$BYTESCHA AND 0EFH) = UASCNTL)
  THEN CALL RCV$UA(1,PSBIT$A); /*RRR FRAME*/
ELSE IF ((CONTROL$BYTESCHA AND 0FH) = RR$CNTL)
  THEN CALL RCV$RR(1,PSBIT$A, SEQ$NUM$A, ADDRESS$A);
ELSE IF ((CONTROL$BYTESCHA AND 0FH) = RNR$CNTL) /*RNR FRAME*/
  THEN CALL RCV$RNR(1,PSBIT$A, SEQ$NUM$A, ADDRESS$B);
ELSE IF ((CONTROL$BYTESCHA AND 0FH) = REJ$CNTL) /*REJ FRAME*/
  THEN CALL RCV$REJ(1,PSBIT$A, SEQ$NUM$A, ADDRESS$B);
ELSE IF ((CONTROL$BYTESCHA AND 0EFH) = SABM$CNTL) /*SABM FRAME*/
  THEN CALL RCV$SABM(1,PSBIT$A);
ELSE IF ((CONTROL$BYTESCHA AND 0EFH) = DISC$CNTL) /*DISC FRAME*/
  THEN CALL RCV$DISC(1,PSBIT$A);
ELSE IF ((CONTROL$BYTESCHA AND 0EFH) = CMDR$CNTL) /*CMDR FRAME*/
  THEN CALL RCV$CMDR(1,PSBIT$A);
ELSE IF ((CONTROL$BYTESCHA AND 0EFH) = DM$CNTL) /*DM FRAME*/
  THEN CALL RCV$DM(1,PSBIT$A);
ELSE
  DO; /* IGNORE INVALID CONTROL FIELD FRAMES */
  CALL SEND$SEQ(@TP$65, LENGTH(TP$65));
  CALL SRVC$TAB$H$SKP(1, I$FRAME$SIZE);
END;
IF NOT SABM$MODE$A THEN
  CALL SEND$SEQ(@CR,LF,'ROUTE$IN NOT IN SABM$MODE$A'), 29);
/* CHANNEL A PROCESSING */
END;
ELSE
  DO; /* IGNORE INVALID FRAMES */
  CALL SEND$SEQ(@TP$65A, LENGTH(TP$65A));
  CALL SRVC$TAB$H$SKP(1, I$FRAME$SIZE);
END;

/*-----CHANNEL B PROCESSING -----*/
CALL SEND$SEQ(@CR,LF,2);
CALL SEND$SEQ(@('ROUTE$IN CHB NS & NE ='), 22);
CALL PRINTI(NT02NS);
CALL PRINTI(NT02NE);
CALL SEND$SEQ(@CR,LF,2);
ADDRESS$B = NT02RX(NT02NS);
IF ((NT02NE - NT02NS) >= I$FRAME$SIZE) OR (NT02NS > NT02NE)) THEN
  IF (ADDRESS$B = $AADD) OR (ADDRESS$B = $BADD) THEN
    DO;
    /******FRAME IS PRESENT IN RECEIVE TABLE*****//
    CALL SEND$SEQ(@TP$66C, LENGTH(TP$66C));
    CONTROL$BYTE$CHB = NT02RX(NT02NS + 1);
    SEQ$NUM$B = SHR((CONTROL$BYTE$CHB AND 0E0H), 5);
    PSBIT$B = SHR((CONTROL$BYTE$CHB AND 010H), 4);

```

SOURCE CODE FOR NETX25.SRC (MAIN MODULE), 18 NOV 85

```

      CALL DPLY$FRAME$HOR(2);
      IF NOT SABM$MODE$B THEN
DO,  IF ((CONTROL$BYTES$CHB AND SABM$CNTL) = SABM$CNTL)
      THEN CALL RCV$SABM(2,1);
      ELSE CALL SND$DM(2,1);

END;
ELSE IF ((CONTROL$BYTES$CHB AND 01H) = 0) THEN /*I FRAME*/
      CALL RCV$I$FRAME(2,$BIT$B);
      ELSE IF ((CONTROL$BYTES$CHB AND 0FH) = RR$CNTL) /*RR FRAME*/
      THEN CALL RCV$RR(2,$BIT$B, SEQ$NUM$B, ADDRESS$B);
      ELSE IF ((CONTROL$BYTES$CHB AND 0FH) = RNR$CNTL) /*RNR FRAME*/
      THEN CALL RCV$RNR(2,$BIT$B, SEQ$NUM$B, ADDRESS$B);
      ELSE IF ((CONTROL$BYTES$CHB AND 0FH) = REJ$CNTL) /*REJ FRAME*/
      THEN CALL RCV$REJ(2,$BIT$B, SEQ$NUM$B, ADDRESS$B);
      ELSE IF ((CONTROL$BYTES$CHB AND 0EFH) = SABM$CNTL) /*SABM FRAME*/
      THEN CALL RCV$SABM(2,$BIT$B);
      ELSE IF ((CONTROL$BYTES$CHB AND 0EFH) = DISC$CNTL) /*DISC FRAME*/
      THEN CALL RCV$DISC(2,$BIT$B);
      ELSE IF ((CONTROL$BYTES$CHB AND 0EFH) = UA$CNTL) /*UA FRAME*/
      THEN CALL RCV$UA(2,$BIT$B);
      ELSE IF ((CONTROL$BYTES$CHB AND 0EFH) = CMDR$CNTL) /*CMDR FRAME*/
      THEN CALL RCV$CMDR(2,$BIT$B);
      ELSE IF ((CONTROL$BYTES$CHB AND 0EFH) = DM$CNTL) /*DM FRAME*/
      THEN CALL RCV$DM(2,$BIT$B);
      ELSE
DO;      CALL SEND$SEQ(@TP$66, LENGTH(TP$66));
      CALL SRVC$TAB$H$SKP(2, I$FRAME$SIZE);
END;
      IF NOT SABM$MODE$B THEN
      END; /* CHANNEL B PROCESSING */
DO;      /* IGNORE INVALID FRAMES */
      CALL SEND$SEQ(@TP$66A, LENGTH(TP$66A));
      CALL SRVC$TAB$H$SKP(2, I$FRAME$SIZE);
END;
      IF (SABM$MODE$A AND SABM$MODE$B) THEN CALL ROUTE$SPACKET;

END ROUTE$IN;

/*****
* DATE: 30 SEP 84
* VERSION: 1.0
* NAME: READ$LINE
* MODULE NUMBER: 7.0
* DESCRIPTION: This procedure reads the input data
* from the ISIS OS. This procedure is used for simu-
* purposes only.
*****/

```

SOURCE CODE FOR NETX25.SRC (MAIN MODULE), 18 NOV 85

```

* PASSED VARIABLES:      None
* RETURNS:              None
* GLOBAL VARIABLES USED: BUFFER, MSG1, MSG4, MSG3, ACTUAL, R$CONN
* GLOBAL VARIABLES CHANGED: STATUS, LOOP, MAC$NETWORK $CODE
* MODULES CALLED:        BUFFER, ACTUAL, LOOP
                        SENDSEQ, ERR$CHK, DQ$READ, ASC$HEX,
                        LOAD
* CALLING MODULES:        main, INFO$XFER
* AUTHOR:      C.T. Childress 30 SEP 84
* HISTORY:     1.0 Capt C.T. Childress - original translated PL/M
* ***** version *****
READ$LINE: PROCEDURE;
*****
DECLARE INDEX WORD;
DECLARE LOOP WORD;

INDEX = 0;

CALL SENDSEQ(@MSG1, LENGTH(MSG1));
ACTUAL = DQ$READ(R$CONN, @BUFFER, 128, @STATUS);
CALL ERR$CHK;

IF (BUFFER(0) = 'Y') OR (BUFFER(0) = 'y') THEN DO;
  CALL SENDSEQ(@MSG4, LENGTH(MSG4));
  ACTUAL = DQ$READ(R$CONN, @BUFFER, 128, @STATUS);
  CALL ERR$CHK;
  LOOP = ASC$HEX(BUFFER(0));
  IF (LOOP >= 1) AND (LOW(LOOP) <= MAX$NETWORK$CODE) THEN DO;
    CALL SENDSEQ(@MSG3, LENGTH(MSG3));
    ACTUAL = DQ$READ(R$CONN, @BUFFER, 128, @STATUS);
    CALL ERR$CHK;
    IF (BUFFER(0) >= '1') AND (BUFFER(0) <= '9') THEN
      DO INDEX = 1 TO ASC$HEX(BUFFER(0));
      CALL LOAD(LOOP);
    END;
  END;
END;

END;

/** ASK FOR ANY CONTROL FRAMES SENT **/
CALL SENDSEQ(@MSG5, LENGTH(MSG5));
ACTUAL = DQ$READ(R$CONN, @BUFFER, 128, @STATUS);
CALL ERR$CHK;

IF (BUFFER(0) = 'Y') OR (BUFFER(0) = 'y') THEN DO;
  CALL SENDSEQ(@MSG6, LENGTH(MSG6));
  ACTUAL = DQ$READ(R$CONN, @BUFFER, 128, @STATUS);
  CALL ERR$CHK;
  LOOP = ASC$HEX(BUFFER(0));
  IF LOOP <= 7 THEN

```

SOURCE CODE FOR NETX25.SRC (MAIN MODULE), 18 NOV 85

```

DO CASE LOOP;
DO;
    /* CASE 0
    CALL SND$CMR(1,1,1,1);
    CALL SND$CMR(2,1,1,1);
    /* CASE 1
    END;
DO;
    /* CASE 1
    CALL SND$RR(1,1,B$ADD);
    CALL SND$RR(2,1,A$ADD);
    /* CASE 1
    END;
DO;
    /* CASE 2
    CALL SND$RNR(1,1,B$ADD);
    CALL SND$RNR(2,1,A$ADD);
    /* CASE 2
    END;
DO;
    /* CASE 3
    CALL SND$REJ(1,1,B$ADD);
    CALL SND$REJ(2,1,A$ADD);
    /* CASE 3
    END;
DO;
    /* CASE 4
    CALL SND$UA(1,1);
    CALL SND$UA(2,1);
    /* CASE 4
    END;
DO;
    /* CASE 5
    CALL SND$DISC(1,1);
    CALL SND$DISC(2,1);
    /* CASE 5
    END;
DO;
    /* CASE 6
    CALL SND$DM(1,1);
    CALL SND$DM(2,1);
    /* CASE 6
    END;
DO;
    /* CASE 7
    CALL SND$SABM(1,1);
    CALL SND$SABM(2,1);
    /* CASE 7
    END;
END;
/* END CASE
/* ASK FOR CONTROL FRAME*/

END;

CALL SENDSEQ(@MSG2, LENGTH(MSG2));
ACTUAL = DQ$READ(R$CONN, @BUFFER, 128, @STATUS);
CALL ERR$CHK;

IF (BUFFER(0) = 'Y') OR (BUFFER(0) = 'y') THEN
    FOREVER = FALSE;
ELSE
    FOREVER = TRUE;

END READ$LINE;

/*****
* DATE: 28 AUG 85
* VERSION: 1.0
*****/

```

SOURCE CODE FOR NETX25.SRC (MAIN MODULE), 18 NOV 85

```

* NAME: START$DM$MODE
* MODULE NUMBER: 8.0
* DESCRIPTION: This procedure initializes both channels
*              to the SABM mode. The UNIDS will stay in the DM state
*              until both channels have received SABM commands.
* PASSED VARIABLES:
* RETURNS:
* GLOBAL VARIABLES USED:
* GLOBAL VARIABLES CHANGED:
* MODULES CALLED: RCV$SABM, SND$SABM, RCV$UA, SND$DM,
                  ROUTE$OUT
* CALLING MODULES: main
* AUTHOR: Mark Weber
* HISTORY: 1.0 Mark Weber- 28 AUG 85-original PL/M version
*****
START$DM$MODE: PROCEDURE;
    DECLARE (CONTROL$BYTE$CHA, CONTROL$BYTE$CHB) BYTE;
    DECLARE (P$BIT$A, P$BIT$B) BYTE;
    DECLARE (SEQ$NUM$A, SEQ$NUM$B) BYTE;
    DECLARE TP$62(*) BYTE DATA(CR,LF,
                                ENTERING THE 'DM MODE',CR,LF);
    DECLARE TP$62A(*) BYTE DATA(CR,LF,
                                IN SABM$MODE$A',CR,LF);
    DECLARE TP$62B(*) BYTE DATA(CR,LF,
                                IN SABM$MODE$B',CR,LF);
    SABM$MODE$A = FALSE;
    SABM$MODE$B = FALSE;
    TX01NS, TX01NE, TX02NS, TX02NE = 0;
    NTO1NS, NTO1NE, NTO2NS, NTO2NE = 0;
    RCV$STATE$A, RCV$STATE$B = 0;
    SEND$STATE$A, SEND$STATE$B = 0;
    CALL READLINE;
    CALL SENDSEQ(@TP$62, LENGTH(TP$62));
    DO WHILE NOT (SABM$MODE$A AND SABM$MODE$B);
        IF ((NTO1NE - NTO1NS) >= 1$FRAME$SIZE) OR (NTO1NS > NTO1NE) THEN
            DO;
                /*****FRAME IS PRESENT IN RECEIVE TABLE*****/
                CONTROL$BYTE$CHA = NTO1RX(NTO1NS + 1);
                SEQ$NUM$A = SHL((CONTROL$BYTE$CHA AND 00FH), 5);
                P$BIT$A = CONTROL$BYTE$CHA AND 010H;
                IF NOT SABM$MODE$A THEN
                    DO;
                        /* CHECK FOR SABM COMMAND*/
                        IF ((CONTROL$BYTE$CHA AND SABM$CNTL) = SABM$CNTL)
                            THEN CALL RCV$SABM(1,1);
                        ELSE IF ((CONTROL$BYTE$CHA AND UA$CNTL) = UA$CNTL) /*UA FRAME*/
                            THEN CALL RCV$UA(1,P$BIT$A);
                        ELSE CALL SND$DM(1,1);
                        /* SND UA RESPONSE*/
                        /*
                    END; /* END IF ELSE ON RECEIVED FRAMES

```


SOURCE CODE FOR NETX25.SRC (MAIN MODULE), 18 NOV 85

```

END; /* END RECEIVED FRAME PROCEDURES FOR CHANNEL A */
/* IF NO FRAME RECEIVED, SEND A SABM COMMAND */
ELSE CALL SND$SABM(1,1);

/*****CHANNEL B PROCESSING *****/
IF((NT02NE - NT02NS) >= I$FRAME$SIZE) OR (NT02NS > NT02NE) THEN
DO;
/*****FRAME IS PRESENT IN RECEIVE TABLE*****/
CONTROL$BYTES$CHB = NT02RX(NT02NS + 1);
SEQ$NUM$B = SHL((CONTROL$BYTES$CHB AND 0EOH), 5);
PS$BIT$B = CONTROL$BYTES$CHB AND 010H;
IF NOT SABM$MODE$B THEN /* CHECK FOR SABM COMMAND*/
DO;
IF ((CONTROL$BYTES$CHB AND SABM$CNTL) = SABM$CNTL)
THEN CALL RCV$SABM(2,1);
ELSE IF ((CONTROL$BYTES$CHB AND UA$CNTL) = UA$CNTL) /*UA FRAME*/
THEN CALL RCV$UA(2,PS$BIT$A); /* SMD UA RESPONSE*/
ELSE CALL SND$DM(2,1); /* END IF ELSE ON RECEIVED FRAMES */
END; /* END RECEIVED FRAME PROCEDURES FOR CHANNEL A */
/* IF NO FRAME RECEIVED, SEND A SABM COMMAND */
ELSE CALL SND$SABM(2,1);
IF SABM$MODE$A THEN CALL SENDSEQ(@TP$62A, LENGTH(TP$62A));
IF SABM$MODE$B THEN CALL SENDSEQ(@TP$62B, LENGTH(TP$62B));

/*****CHECK FOR ANY FRAMES SENT*****/
CALL ROUTE$OUT;
END; /*END WHILE */
END START$DM$MODE;

/*****
* DATE: 28 AUG 85
* VERSION: 1.0
* NAME: START$INFO$XFER
* MODULE NUMBER: 9.0
* DESCRIPTION: This is the driver module for the inform-
* action transfer phase (state p4 of X.25). This module
* calls the receive and transmit portions of the LAP B
* protocol currently implemented.
* PASSED VARIABLES: None
* RETURNS: None
* GLOBAL VARIABLES USED: SABM$MODE
* GLOBAL VARIABLES CHANGED: SABM$MODE
* MODULES CALLED: ROUTE$IN, ROUTE$OUT
* CALLING MODULES: main
* AUTHOR: Mark Weber
* HISTORY: 1.0 Mark Weber - 28 AUG 85 - original PL/M version
*****/
START$INFO$XFER; PROCEDURE;

```

SOURCE CODE FOR NETX25.SRC (MAIN MODULE), 18 NOV 85

```

DECLARE TP$63(*) BYTE DATA(CR,LF,
    'TP$63
    Entering information transfer state (p4 of X.25)');

RNR$MODE$A = FALSE;
RNR$MODE$B = FALSE;
CALL SENDSEQ(@TP$63, LENGTH(TP$63));
DO WHILE ((SABM$MODE$A AND SABM$MODE$B) AND FOREVER);
    CALL ROUTE$IN;
    CALL ROUTE$OUT;
    CALL READ$TAB;
    CALL SENDSEQ(@TP$43, LENGTH(TP$43));
    CALL READ$LINE; /* END ONE SERVICE OF TRANSMIT AND RECEIVE */
END;

```

END START\$INFO\$XFER;

```

/*****
* DATE: 16 SEP 85
* VERSION: 1.1
* NAME: main
* MODULE NUMBER: None
* DESCRIPTION: This is the main module for the data-
  link layer simulation test. The main module acts as
  as an executive procedure, calling the primary pro-
  cedures implementing the LAP B protocol. Of primary
  concern are the modules READ$LINE, START$DM$MODE, and
  START$INFO$XFER. READ$LINE evaluates user responses
  and loads the proper table with correctly formatted
  frames or packets. START$DM$MODE simulates the initial-
  ization for the SABM mode operation. START$INFO$XFER
  calls the primary SABM mode routines ROUTE$IN and
  ROUTE$OUT for information transfer in the SABM mode.
  PASSED VARIABLES:
  RETURNS:
  GLOBAL VARIABLES USED:
  GLOBAL VARIABLES CHANGED:
  MODULES CALLED:
  DQ$CREATE, SENDSEQ, INIT$NSTAB, READ$LINE,
  START$DM$MODE, START$INFO$XFER, DQ$DETACH, DQ$EXIT
  CALLING MODULES:
  ALGORITHM: MAIN: OPEN write device (system console)
  OPEN read device (system console)
  OPEN file output (FILE.OUT)
  Send start up header
  Send initialization message
  Initialize global variables, tables,
  and pointers
  Ask to continue the test
  Read response variable FOREVER
  Send system start message
  */

```

MODULE FOR NETX25.SRC (MAIN MODULE), 18 NOV 85

```

*
* While FOREVER true
*   Send inside FOREVER loop message
*   Start DM Mode
*   Send exited DM Moded message
*   Start Information Transfer Phase
*   Send exited Information Transfer Phase
*   END while loop
*   Send end of test message
*   Close output file (FILE.OUT)
*   Close write device
*   Close read device
*
* END MAIN;
* AUTHOR: 1.1 Mark Weber - 16 SEP 85 - LAP B datalink implementation*
* HISTORY: 1.0 Capt C.T. Chidress - original translated PL/M
*
*****

```

BEGIN:

```

/* THE FOLLOWING DQ$xxxxx CALLS MUST BE THE FIRST EXECUTABLE CODE IN
THIS MODULE FOR THE ISIS-II CONSOLE I/O TO OPERATE PROPERLY.
DO NOT CHANGE THE LOCATION OF THIS CODE OR IT WILL NOT WORK!
DO NOT MAKE 'BEGIN;' PUBLIC AND THE START(BEGIN) OPTION WITH
LINK86 OR IT WILL NOT WORK!
*/

```

```

W$CONN = DQ$CREATE (@(4, ':CO:'), @STATUS);
CALL ERR$CHK;
CALL DQ$OPEN (W$CONN, 2, 0, @STATUS);
CALL ERR$CHK;

R$CONN = DQ$ATTACH (@(4, ':CI:'), @STATUS);
CALL ERR$CHK;
CALL DQ$OPEN (R$CONN, 1, 0, @STATUS);
CALL ERR$CHK;

F$CONN = DQ$CREATE (@(8, 'FILE.OUT'), @STATUS);
CALL ERR$CHK;
CALL DQ$OPEN (F$CONN, 2, 1, @STATUS);
CALL ERR$CHK;

CALL SENDSEQ(@STARTUP$HDR, LENGTH(STARTUP$HDR));

CALL SENDSEQ(@TP$1, LENGTH(TP$1));

CALL INIT$N$TAB;

CALL SENDSEQ(@TP$2, LENGTH(TP$2));

CALL READ$LINE;

```

SOURCE CODE FOR NETX25.SRC (MAIN MODULE). 18 NOV 85

```
CALL SENDSEQ(@TP$3, LENGTH(TP$3));
DO WHILE FOREVER;
  CALL SENDSEQ(@TP$4, LENGTH(TP$4));
  CALL STARTSDM$MODE;          /* BOTH CHANNELS ARE NOW IN SABM$MODE */
  CALL SENDSEQ(@TP$42, LENGTH(TP$42));
  CALL START$INFO$XFER;
END;
CALL SENDSEQ(@TP$44, LENGTH(TP$44));
CALL DQ$CLOSE(F$CONN, @STATUS);
CALL ERR$CHK;
CALL DQ$CLOSE(W$CONN, @STATUS);
CALL ERR$CHK;
CALL DQ$CLOSE(R$CONN, @STATUS);
CALL ERR$CHK;
CALL DQ$DETACH(F$CONN, @STATUS);
CALL ERR$CHK;
CALL DQ$DETACH(W$CONN, @STATUS);
CALL ERR$CHK;
CALL DQ$DETACH(R$CONN, @STATUS);
CALL ERR$CHK;
CALL DQ$EXIT(0);

END MAIN;      /* END MAIN MODULE */
/***** THE END *****/
```

2. LAPB0.SRC - Module with LAP B procedures

SOURCE CODE FOR LAPB0.SRC MODULE TO NETX.25, 5 NOV 85

```

$TITLE('UNID II NETWORK TEST PROGRAM, 5 NOV 85')
$XREF OPTIMIZE(2)
/*****
* DATE: 6 NOV 85
* VERSION: 1.0
* TITLE: ISO layer 2 LAP B simulation
* FILENAME: LAPB0.SRC
* COORDINATOR: Capt Mark W. Weber
* PROJECT: UNID II
* OPERATING SYSTEM: INTEL SYSTEM III/230
* LANGUAGE: PL/M 86
* USE: This file requires no includes. Compliation is with the compact
* and other default vaules of the PLM86 compiler. Linking requires
* the main module, NETX25.OBJ, the module LAPB1.OBJ, and SMALL.LIB.
* Use the SUBMIT file NETX25.CSD for locating and linking the object
* code.
* CONTENTS: LAPB - module containing ISO layer two procedures
* PRINTI - prints an integer given the hex input
* DSPLY$FRAME$HDR - displays the frame header and prints the
* pointer locations in the transmit and
* receive tables
* STOS$CMD - stores the control byte given in a FIFO queue
* FIND$CMD - finds the next U command to service
* UPDATES$END$STATE - updates the send state variable
* SDC$CMDR - places a CMDR formatted frame in the transmt queue
* RESET$QUE - places the next to process pointer at specified
* location to retransmit a frame
* SDC$DM - places a DM formatted response in the transmit queue
* SDC$SABM - places a SABM fomatted frame in the transmit queue
* SDC$IN$SEQ - checks for in sequence send state variables
* RCV$IN$SEQ - checks for in sequence receive state variables
* SDC$RR - places a RR frame in the transmit queue
* SDC$REJ - places a REJ frame in the transmit queue
* SDC$RNR - places a RNR frame in the transmit queue
* SDC$UA - places a UA frame in the transmit queue
* SDC$DISC - places a DISC frame in the transmit queue
* RCV$I$FRAME - processes received I$FRAMES, and is entry point
* point for ISO layer 3 receive packet software
* RCV$SABM - processes received SABM frames
* RCV$UA - processes received UA frames
* RCV$REJ - processes received REJ frames
* RCV$RNR - processes received RNR frames
* RCV$RR - processes received RR frames
* RCV$DICS - processes received DICS frames
* RCV$CMDR - processes received CMDR frames
* FRAME$PRESENT - checks for a frame in the datalink receive
* table
* FUNCTION: Simulates the operation of the SBC 88/45 in the UNID II using
* the datalink layer as described by CCITT recommendation X.25.
* ISIS calls are used to provide a user interface and to trace
  
```

SOURCE CODE FOR LAPB0.SRC MODULE TO NETX.25, 5 NOV 85

```
*
* the process flow of the LAP B protocol employed by the CCITT
* X.25 recommendation.
* HISTORY: 1.0 Mark Weber - 7 SEP 84 - original version
*****
```

```
LAPB$MODULE: DO;
/***** EXTERNAL PROCEDURES FOR ISIS SYSTEM CALLS *****/
```

```
LD$TAB$H$SKP: PROCEDURE (TABLE, FRAME$SIZE) EXTERNAL;
DECLARE TABLE BYTE, FRAME$SIZE INTEGER;
END LD$TAB$H$SKP;
```

```
SRVC$TAB$H$SKP: PROCEDURE (TABLE, FRAME$SIZE) EXTERNAL;
DECLARE TABLE BYTE, FRAME$SIZE INTEGER;
END SRVC$TAB$H$SKP;
```

```
SENDSEQ: PROCEDURE (MSG, TOTAL) EXTERNAL;
DECLARE MSG POINTER, TOTAL WORD;
END SENDSEQ;
```

```
DET$DEST$LN: PROCEDURE WORD EXTERNAL;
END DET$DEST$LN;
```

```
DET$DEST$ONE: PROCEDURE WORD EXTERNAL;
END DET$DEST$ONE;
```

```
DET$DEST$TWO: PROCEDURE WORD EXTERNAL;
END DET$DEST$TWO;
```

```
HEX$ASC: PROCEDURE (C) WORD EXTERNAL;
DECLARE C BYTE;
END HEX$ASC;
```

```
INITACK: PROCEDURE (TABLE) EXTERNAL;
DECLARE TABLE BYTE;
END INITACK;
```

```
DECLARE CR LITERALLY 'ODH',
LF LITERALLY 'OAH';
```

```
/***** END EXTERNALS *****/
```

```
DECLARE FALSE LITERALLY 'OH', /* USE AS FLAGS TO TEST */
TRUE LITERALLY 'OFFH', /* BITS FOR BRANCHING */
CONCTC LITERALLY 'ODSH', /* NETWORK MONITOR CTC PORT ADDRESS */
CONCMD LITERALLY 'ODFH';
```

```

CONDAT      LITERALLY 'ODEH',
/* NETWORK MONITOR USART COMMAND PORT ADDRESS */
NET$R1$DEST$ERR LITERALLY '10', /* NETWORK MONITOR USART DATA PORT ADDRESS */
NET$R0$DEST$ERR LITERALLY '11', /* NET ROUTE$IN DEST ERROR ENTRY */
PACKET$SIZE LITERALLY '138', /* NET ROUTE$OUT DEST ERROR ENTRY */
PACKET$IN$TABLE LITERALLY '10', /* PACKET IS 138 BYTE BLOCK */
PACKET$TABLE$SIZE LITERALLY '1380', /* # PACKETS IN TABLE */
I$FRAME$SIZE LITERALLY '140',
US$FRAME$SIZE LITERALLY '2',
SS$FRAME$SIZE LITERALLY '2',
CMDR$FRAME$SIZE LITERALLY '5',
FRAME$IN$TABLE LITERALLY '10',
FRAME$TABLE$SIZE LITERALLY '1400',
DATA$SIZE LITERALLY '128',
IP$DATA$SIZE LITERALLY '96',
TCP$DATA$SIZE LITERALLY '72';

```

```
/* NETWORK VARIABLES FOR THIS UNID */
```

```

/* NOTES: 1. THIS$UNID$NBR MUST REFLECT WHICH UNID THIS IS.
2. THIS$COUNTRY$CODE MUST REFLECT THE AREA TO WHICH THIS
   UNID IS LOCATED.
3. MAX$COUNTRY$CODE WILL INDICATE WHICH COUNTRY CODES
   ARE CURRENTLY OPERATIONAL. CC = 0000 IS RESERVED
   FOR THE DELNET MONITOR.
4. MAX$NETWORK$CODE WILL INDICATE HOW MANY UNIDS ARE
   CURRENTLY OPERATIONAL WITHIN A PARTICULAR COUNTRY.
5. FOR DETAILED INFORMATION ON THE ABOVE, REFER TO
   PHISTER'S THESIS, APPENDIX D.
*/

```

```
DECLARE
```

```

THIS$UNID$NBR LITERALLY '02H', /* UNIQUE ADDRESS OF THIS UNID */
THIS$COUNTRY$CODE LITERALLY '01H', /* COUNTRY WHERE THIS UNID RESIDES */
MAX$COUNTRY$CODE LITERALLY '01H', /* COUNTRIES CURRENTLY OPERATIONAL */
MAX$NETWORK$CODE LITERALLY '03H', /* NUMBER OF UNIDS OPERATIONAL IN CC */

```

```

STAT$NBR LITERALLY '20', /* NUMBER OF ENTRIES IN STATUS TABLE */

/* VARIABLES USED IN N.MAIN$U2 AND N.INSIO$U2 */
CTCNOA BYTE,
/* PROGRESSIVE NUMBER OF TIME COUNTS FOR NETWORK CHANNEL A */
CTCNOB BYTE,
/* PROGRESSIVE NUMBER OF TIME COUNTS FOR NETWORK CHANNEL B */
MAXNOA LITERALLY '3', /* PREVIOUSLY 64H = 1000 */
/* MAXIMUM NUMBER OF TIMING COUNTS FOR NETWORK CHANNEL A */
MAXNOB LITERALLY '3', /* PREVIOUSLY 64H = 1000 */
/* MAXIMUM NUMBER OF TIMING COUNTS FOR NETWORK CHANNEL B */

```


SOURCE CODE FOR LAPB0.SRC MODULE TO NETX.25, 5 NOV 85

```

RETRANSSA  BYTE
/* PROGRESSIVE NUMBER OF RETRANSMISSIONS OF A FRAME */
RETRANSB   BYTE
/* PROGRESSIVE NUMBER OF RETRANSMISSIONS OF A FRAME */
MAXRETRANSSA  LITERALLY '6',
/* MAXIMUM NUMBER OF RETRANSMISSIONS OF A FRAME */
MAXRETRANSB   LITERALLY '6',
/* MAXIMUM NUMBER OF RETRANSMISSIONS OF A FRAME */

DECLARE BUSYSTATUS LITERALLY 'OFFH',
NMBS$MSK          LITERALLY '07H',
ESC               LITERALLY '1BH',
EOT               LITERALLY '04H',

DECLARE ASCII(16)  BYTE EXTERNAL;
DECLARE TEMP(24)   WORD EXTERNAL;
DECLARE MSGNUM     BYTE EXTERNAL;
DECLARE OUT$TAB$FULL BYTE EXTERNAL;
DECLARE A$ADD      LITERALLY '0000$0001B', /*LAP B ADDRESS BYTE FORMATS */
B$ADD              LITERALLY '0000$0001B';

DECLARE TEST      LITERALLY '0';
/* 0 - OPTIONAL PRINT STATEMENTS
NOT MADE
*/
/*LAP B FRAME CONTROL BYTE FORMATS*/

DECLARE I$CNTL    LITERALLY '0000$0001B',
RR$CNTL          LITERALLY '0000$0001B',
RNR$CNTL         LITERALLY '0000$0101B',
REJ$CNTL         LITERALLY '0000$1001B',
DM$CNTL          LITERALLY '0000$1111B',
SABM$CNTL        LITERALLY '0010$1111B',
DISC$CNTL        LITERALLY '0100$0011B',
UA$CNTL          LITERALLY '0110$0011B',
CMDS$CNTL        LITERALLY '1000$0111B',
P$BIT$MASK       LITERALLY '010H';

/****** GLOBAL CONSTANTS USED BY LAP B PROCESSES *****/

DECLARE (SABM$MODE$A, SABM$MODE$B)  BYTE EXTERNAL;
DECLARE (RNR$MODE$A, RNR$MODE$B)    BYTE EXTERNAL;
DECLARE (UA$ACK$CHA, UA$ACK$CHB)     BYTE EXTERNAL;
DECLARE DM$MODE                      BYTE EXTERNAL;
DECLARE US$CMD$QUE$A STRUCTURE(ACK BYTE, CMD BYTE) EXTERNAL;
DECLARE US$CMD$QUE$B STRUCTURE(ACK BYTE, CMD BYTE) EXTERNAL;

/* MORE TO BE ADDED LATER */

/***** ADDITIONAL GENERAL DECLARES NEEDED FOR THIS PROGRAM *****/
/*****
DECLARE TIMCHA          BYTE EXTERNAL,
TIMCHB                 BYTE EXTERNAL,

```

SOURCE CODE FOR LAPBO.SRC MODULE TO NETX.25, 5 NOV 85

```

      HSKP$ERR      INTEGER EXTERNAL;
      /* MORE TO BE ADDED LATER */
      /*****
      /* DATA TABLES USED IN THIS PROGRAM
      /*****

```

DECLARE

```

      NT01RX(FRAME$TABLE$SIZE)  BYTE EXTERNAL,
      NT01NS  INTEGER EXTERNAL,
      NT01NE  INTEGER EXTERNAL,
      NT01SZ  INTEGER EXTERNAL,

```

```

      NT02RX(FRAME$TABLE$SIZE)  BYTE EXTERNAL,
      NT02NS  INTEGER EXTERNAL,
      NT02NE  INTEGER EXTERNAL,
      NT02SZ  INTEGER EXTERNAL,

```

```

      NT01TX(FRAME$TABLE$SIZE)  BYTE EXTERNAL,
      TX01NS  INTEGER EXTERNAL,
      TX01NE  INTEGER EXTERNAL,
      TX01SZ  INTEGER EXTERNAL,

```

```

      NT02TX(FRAME$TABLE$SIZE)  BYTE EXTERNAL,
      TX02NS  INTEGER EXTERNAL,
      TX02NE  INTEGER EXTERNAL,
      TX02SZ  INTEGER EXTERNAL,

```

```

      I$FRAME$QUEUE(FRAME$TABLE$SIZE)  BYTE EXTERNAL,
      I$FRAME$QUEUE$NS  INTEGER EXTERNAL,
      I$FRAME$QUEUE$NE  INTEGER EXTERNAL,
      I$FRAME$QUEUE$SZ  INTEGER EXTERNAL,

```

```

      LCNTTB(PACKET$TABLE$SIZE)  BYTE EXTERNAL,
      LCNTNS  INTEGER EXTERNAL,
      LCNTNE  INTEGER EXTERNAL,
      LCNTSZ  INTEGER EXTERNAL,

```

```

      NTLCTB(PACKET$TABLE$SIZE)  BYTE EXTERNAL,
      NTL CNS  INTEGER EXTERNAL,
      NTL CNE  INTEGER EXTERNAL,
      NTL CSZ  INTEGER EXTERNAL,

```

```

      STAT$B(STAT$NBR)  BYTE;

```

/* MISCELLANEOUS DECLARATIONS */

DECLARE

```

      FOREVER BYTE EXTERNAL,
      DESTINATION WORD EXTERNAL,
      /* DESTINATION OF A FRAME */

```

SOURCE CODE FOR LAP80.SRC MODULE TO NETX.25, 5 NOV 85

/* ACKNOWLEDGE VARIABLE DECLARATIONS */

```

SEQ$NUM$A  BYTE EXTERNAL,
SEQ$NUM$B  BYTE EXTERNAL,
SEND$STATE$A  BYTE EXTERNAL,
SEND$STATE$B  BYTE EXTERNAL,
RCV$STATE$A  BYTE EXTERNAL,
RCV$STATE$B  BYTE EXTERNAL;

```

/* MESSAGES SENT TO MONITOR PORT */

```

DECLARE TP$75(*) BYTE DATA(CR,LF,
'TP$75 ERROR IN RCV$CMDR CHANNEL NUMBER');
DECLARE TP$75A(*) BYTE DATA(CR,LF,
'TP$75A ENTERING RCV$CMDR, CHANNEL A');
DECLARE TP$75B(*) BYTE DATA(CR,LF,
'TP$75B ENTERING RCV$CMDR, CHANNEL B');

```

```

/*****
* DATE: 11 SEP 85
* VERSION: 1.0
* NAME: PRINTI
* MODULE NUMBER: 8.3.2.2
* DESCRIPTION: This procedure prints integers
* passed to it. It first converts the integer to ASCII
* then it calls SENDSEQ to print the ASCII string.
* PASSED VARIABLES: NUM
* RETURNS: None
* GLOBAL VARIABLES USED: TEMP1
* GLOBAL VARIABLES CHANGED: TEMP1
* MODULES CALLED: SEND$SEQ
* CALLING MODULES: DSPLY$FRAME$HDR
* AUTHOR: Mark Weber
* HISTORY: 1.0 Mark Weber
*****
PRINTI: PROCEDURE (NUM) PUBLIC;
DECLARE (NUM,I,X) INTEGER;
DECLARE TEMP1(6) BYTE;
DECLARE TEMP2(6) BYTE;

```

```

DO I = 0 TO 5;
  TEMP1(I), TEMP2(I) = '*';
END;
I=0;
X = NUM MOD 10;
TEMP1(I) = ASCII(X);
DO WHILE NUM > 0;
  I = I + 1;
  NUM = NUM/10;
  X = NUM MOD 10;
  TEMP1(I) = ASCII(X);

```

SOURCE CODE FOR LAPB0.SRC MODULE TO NETX.25, 5 NOV 85

```

END;
DO X = 0 TO I;
  TEMP2(X) = TEMP1(I - X);
END;
CALL SENDSEQ(@TEMP2, 6);
END PRINTI;

/*****
* DATE: 7 SEP 85
* VERSION: 1.0
* NAME: DPLY$FRAME$HDR
* MODULE NUMBER: 9.3.7
* DESCRIPTION: This procedure displays the header bytes
*               in frames ready to be served by ROUTE$OUT and ROUTE$IN
*               and prints the pointer values NE and NS for the
*               specified transmit table.
* PASSED VARIABLES: CHANNEL
* RETURNS: None
* GLOBAL VARIABLES USED: TX01NE, TX01NS, TX02NE, TX02NS, TEMP
*                       NX01NE, NX01NS, NX02NE, NX02NS
* GLOBAL VARIABLES CHANGED: TEMP
* MODULES CALLED: SEND$DATA, HEX$ASC
* CALLING MODULES: ROUTE$OUT
* AUTHOR: Mark Weber
* HISTORY: 1.0 Mark Weber - 7 SEP 85 - original
*****/
DPLY$FRAME$HDR: PROCEDURE(CHANNEL) PUBLIC REENTRANT;

  DECLARE CHANNEL BYTE;
  DECLARE (INDEX, SIZE) INTEGER;
  DECLARE TP$DSP$F$RCVA(*) BYTE DATA(CR,LF);
  DECLARE TP$DSP$F$RCV-A FRAME NOW BEING SERVED', CR,LF);
  DECLARE TP$DSP$F$RCVB(*) BYTE DATA(CR,LF);
  DECLARE TP$DSP$F$RCV-B FRAME NOW BEING SERVED', CR,LF);
  DECLARE TP$DSP$F$SND-A(*) BYTE DATA(CR,LF);
  DECLARE TP$DSP$F$SND-A FRAME NOW BEING SERVED', CR,LF);
  DECLARE TP$DSP$F$SND-B(*) BYTE DATA(CR,LF);
  DECLARE TP$DSP$F$SND-B FRAME NOW BEING SERVED', CR,LF);

  DO CASE CHANNEL;
    /* 0 IS A NULL
    ;
    DO;
      /* CASE 1, CHANNEL A
      */
      IF NTO1NS <> NTO1NE THEN
        DO;
          CALL SENDSEQ(@TP$DSP$F$RCVA, LENGTH(TP$DSP$F$RCVA));
          IF ((NTO1RX(NTO1NS + 1) AND I$CNTL) = 0) THEN
            DO INDEX = 0 TO 11;
              TEMP(INDEX) = HEX$ASC(NTO1RX(NTO1NS + INDEX));
            END;
          ELSE

```

```

DO INDEX = 0 TO 11;
  IF INDEX >= 2 THEN
    TEMP(INDEX) = '*';
  ELSE TEMP(INDEX) = HEX$ASC(NT01RX(NT01NS + INDEX));
END;
CALL SENDSEQ(@TEMP, LENGTH(TEMP));
END;
CALL SENDSEQ(@CR,LF, 2);
CALL PRINTI(NT01NS);
CALL PRINTI(NT01NE);
END;
/* CASE 1
/* CASE 2
DO;
  IF NT02NS <> NT02NE THEN
DO;
  CALL SENDSEQ(@TP$DSP$F$RCVB, LENGTH(TP$DSP$F$RCVB));
  IF ((NT02RX(NT02NS + 1) AND I$CNTL) = 0) THEN
DO INDEX = 0 TO 11;
  TEMP(INDEX) = HEX$ASC(NT02RX(NT02NS + INDEX));
END;
ELSE
DO INDEX = 0 TO 11;
  IF INDEX >= 2 THEN
    TEMP(INDEX) = '*';
  ELSE TEMP(INDEX) = HEX$ASC(NT02RX(NT02NS + INDEX));
END;
CALL SENDSEQ(@TEMP, LENGTH(TEMP));
END;
CALL SENDSEQ(@CR,LF, 2);
CALL PRINTI(NT02NS);
CALL PRINTI(NT02NE);
END;
/* CASE 2
/* CASE 3, CHANNEL A
DO;
  IF TX01NS <> TX01NE THEN
DO;
  CALL SENDSEQ(@TP$DSP$F$SND, LENGTH(TP$DSP$F$SND));
  IF ((NT01TX(TX01NS + 1) AND I$CNTL) = 0) THEN
DO INDEX = 0 TO 11;
  TEMP(INDEX) = HEX$ASC(NT01TX(TX01NS + INDEX));
END;
ELSE
DO INDEX = 0 TO 11;
  IF INDEX >= 2 THEN
    TEMP(INDEX) = '*';
  ELSE TEMP(INDEX) = HEX$ASC(NT01TX(TX01NS + INDEX));
END;
CALL SENDSEQ(@TEMP, LENGTH(TEMP));
END;
CALL SENDSEQ(@CR,LF, 2);
CALL PRINTI(TX01NS);
CALL PRINTI(TX01NE);

```

```

END;          /* CASE 3          */
DO;           /* CASE 4          */
  IF TX02NS <> TX02NE THEN
  DO;
    CALL SENDSEQ(@TP$DSP$F$SNDB, LENGTH(TP$DSP$F$SNDB));
    IF ((NT02TX(TX02NS + 1) AND 1$CNTL) = 0) THEN
      DO INDEX = 0 TO 11;
        TEMP(INDEX) = HEX$ASC(NT02TX(TX02NS + INDEX));
      END;
    ELSE
      DO INDEX = 0 TO 11;
        IF INDEX >= 2 THEN
          TEMP(INDEX) = '*';
        ELSE TEMP(INDEX) = HEX$ASC(NT02TX(TX02NS + INDEX));
      END;
    CALL SENDSEQ(@TEMP, LENGTH(TEMP));
  END;
  CALL SENDSEQ(@CR,LF), 2);
  CALL PRINTI(TX02NS);
  CALL PRINTI(TX02NE);
END;          /* CASE 4          */
END D$PLY$FRAME$HOR;

/*****
* DATE:          29 AUG 85
* VERSION:       1.0
* NAME:          FRAME$PRESENT
* MODULE NUMBER: 9.3.1
* DESCRIPTION:   This function determines if a frame is
*               waiting in the transmit buffer for processing. A TRUE
*               value is returned if a frame is found.
* PASSED VARIABLES:
* RETURNS:      CHANNEL
* GLOBAL VARIABLES USED: TX01NE, TX02NE
* GLOBAL VARIABLES CHANGED: None
* MODULES CALLED: SENDSEQ
* CALLING MODULES: ROUTE$OUT
* AUTHOR:       Mark Weber
* HISTORY:      1.0 Mark Weber -29 AUG 85-original PL/M version.
*****/
FRAME$PRESENT: PROCEDURE(CHANNEL) BYTE PUBLIC;

DECLARE CHANNEL BYTE;
DECLARE STATUS BYTE;
DECLARE TP$B9(*) BYTE DATA(CR,LF,
'TP$B9
ENTERING FRAME$PRESENT, CHANNEL A');
DECLARE TP$B9A(*) BYTE DATA(CR,LF,
'TP$B9A
ENTERING FRAME$PRESENT, CHANNEL B');

```

SOURCE CODE FOR LAPBO.SRC MODULE TO NETX.25, 5 NOV 85

```

IF((CHANNEL = 1) OR (CHANNEL = 2)) THEN
DO CASE CHANNEL:
DO:
/* CASE 0 IS NULL */
/* CASE 1 */
IF TEST THEN CALL SENDSEQ(@TP$89, LENGTH(TP$89));
IF( ((TX01NE - TX01NS) >= I$FRAME$SIZE) OR
(TX01NS > TX01NE) ) THEN
STATUS = TRUE;
ELSE STATUS = FALSE;
END; /* CASE 1 */
DO: /* CASE 2 */
IF TEST THEN CALL SENDSEQ(@TP$89A, LENGTH(TP$89A));
IF( ((TX02NE - TX02NS) >= I$FRAME$SIZE) OR
(TX02NS > TX02NE) ) THEN
STATUS = TRUE;
ELSE STATUS = FALSE;
END; /* CASE 2 */
/* END CASE */
RETURN STATUS;
END FRAME$PRESENT;
/*****
* DATE: 23 AUG 85
* VERSION: 1.0
* NAME: STOS$CMD
* MODULE NUMBER: 8.5.2
* DESCRIPTION: This procedure stores all U command
frames in a variable when a command is sent. Receiving
a UA response activates the last U command not
acknowledged.
* PASSED VARIABLES: CHANNEL, CNTL$BYTE
* RETURNS: None
* GLOBAL VARIABLES USED: US$CMD$QUE$A, US$CMD$QUE$A
* GLOBAL VARIABLES CHANGED: all of above
* MODULES CALLED: None
* CALLING MODULES: SND$SABM, SND$DISC, SENDSEQ
* AUTHOR: Mark Weber
* HISTORY: 1.0 Mark Weber
*****
STOS$CMD: PROCEDURE(CHANNEL, CNTL$BYTE) PUBLIC;
DECLARE(CHANNEL, CNTL$BYTE) BYTE;
DECLARE TP$STOS$A(*) BYTE DATA(CR,LF,
'TP$STOS$A ENTERING STOS$CMD, CHANNEL A');
DECLARE TP$STOS$B(*) BYTE DATA(CR,LF,
'TP$STOS$B ENTERING STOS$CMD, CHANNEL B');
IF ((CHANNEL = 1) OR (CHANNEL = 2)) THEN
DO CASE CHANNEL:
; /* CASE 0 IS NULL */

```

```

DO:      /* CASE 1 */
        CALL SENDSEQ(@TP$STOSA, LENGTH(TP$STOSA));
        IF (USCMD$Q$A.ACK = TRUE) THEN
DO:      USCMD$Q$A.ACK = FALSE;
        USCMD$Q$A.CMD = CNTL$BYTE;
END:      /* END CASE 1 */
DO:      /* CASE 2 */
        CALL SENDSEQ(@TP$STOB$, LENGTH(TP$STOB$));
        IF (USCMD$Q$B.ACK = TRUE) THEN
DO:      USCMD$Q$B.ACK = FALSE;
        USCMD$Q$B.CMD = CNTL$BYTE;
END:      /* END CASE 2 */
END;      /* END CASE */
END STOS$CMD;

/*****
DATE:      23 AUG 85
VERSION:    1.0
NAME:       FIND$US$CMD
MODULE NUMBER: 8.3.3
DESCRIPTION: This procedure finds the last
              unacknowledged U command sent. When found the command
              is then processed. CHANNEL
              None
              USCMD$Q$A, USCMD$Q$B$,
              RCV$STATES$, RCV$STATES$, NTO1NE, NTO2NE,
              TXO1NE, TXO1NE
              all of above
              None
              RCV$SABM, RCV$DISC
GLOBAL VARIABLES CHANGED:
MODULES CALLED:
CALLING MODULES:
AUTHOR:     Mark Weber
HISTORY:    1.0 Mark Weber -23 AUG 85
*****
FIND$US$CMD: PROCEDURE(CHANNEL) PUBLIC REENTRANT;
*****
DECLARE CHANNEL BYTE;
DECLARE CNTL$BYTE BYTE;
DECLARE TP$86(*) BYTE DATA(CR,LF,
'TP$86
ERROR IN USCMD$Q$A RETRIVAL');
DECLARE TP$86A(*) BYTE DATA(CR,LF,
'TP$86A
ENTERING FIND$US$CMD CHANNEL A');
DECLARE TP$87(*) BYTE DATA(CR,LF,
'TP$87
ERROR IN USCMD$Q$B RETRIVAL');
DECLARE TP$87A(*) BYTE DATA(CR,LF,
'TP$87A
ENTERING FIND$US$CMD CHANNEL B');
*****/

```


AD-A164 042 DEVELOPMENT AND IMPLEMENTATION OF THE X25 PROTOCOL FOR 3/3
THE UNIVERSAL METH. (U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. M W WEBER
UNCLASSIFIED DEC 85 AFIT/GE/ENG/85D-52-VOL-2 F/G 17/2 NL

AD-A164 042 DEVELOPMENT AND IMPLEMENTATION OF THE X25 PROTOCOL FOR 3/3
THE UNIVERSAL METH. (U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. M W WEBER
UNCLASSIFIED DEC 85 AFIT/GE/ENG/85D-52-VOL-2 F/G 17/2 NL

AD-A164 042 DEVELOPMENT AND IMPLEMENTATION OF THE X25 PROTOCOL FOR 3/3
THE UNIVERSAL METH. (U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. M W WEBER
UNCLASSIFIED DEC 85 AFIT/GE/ENG/85D-52-VOL-2 F/G 17/2 NL

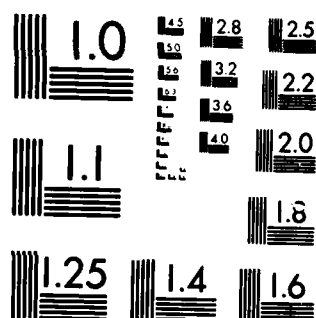
AD-A164 042 DEVELOPMENT AND IMPLEMENTATION OF THE X25 PROTOCOL FOR 3/3
THE UNIVERSAL METH. (U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. M W WEBER
UNCLASSIFIED DEC 85 AFIT/GE/ENG/85D-52-VOL-2 F/G 17/2 NL

AD-A164 042 DEVELOPMENT AND IMPLEMENTATION OF THE X25 PROTOCOL FOR 3/3
THE UNIVERSAL METH. (U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. M W WEBER
UNCLASSIFIED DEC 85 AFIT/GE/ENG/85D-52-VOL-2 F/G 17/2 NL

AD-A164 042 DEVELOPMENT AND IMPLEMENTATION OF THE X25 PROTOCOL FOR 3/3
THE UNIVERSAL METH. (U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. M W WEBER
UNCLASSIFIED DEC 85 AFIT/GE/ENG/85D-52-VOL-2 F/G 17/2 NL

AD-A164 042 DEVELOPMENT AND IMPLEMENTATION OF THE X25 PROTOCOL FOR 3/3
THE UNIVERSAL METH. (U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. M W WEBER
UNCLASSIFIED DEC 85 AFIT/GE/ENG/85D-52-VOL-2 F/G 17/2 NL

[illegible][illegible][illegible]



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A


```

END;
DO:
  SAMSMODE$B = FALSE;
  RNR$MODE$B = FALSE;
  SEND$STATE$B = 0;
  RCV$STATE$B = 0;
  NT02NE, NT02NS = 0;
  TX02NE, TX02NS = 0;
END;

/* CASE 3 IS A UA RESPONSE */
/* CASE 4 IS A CMDR RESPONSE */
/* ERROR OCCURED */
/* ERROR OCCURED */
/* ERROR OCCURED */

CALL SEND$SEQ(@TP$86, LENGTH(TP$86));
CALL SEND$SEQ(@TP$86, LENGTH(TP$86));
CALL SEND$SEQ(@TP$86, LENGTH(TP$86));
END; /* END CMDR CASE */

END;
END; /* END CASE 2 */
/* END CHANNEL CASE */
END FIND$CMD;

/*****
* DATE: 29 AUG 85
* VERSION: 1.0
* NAME: UP$DATE$SEND$STATE
* MODULE NUMBER: 9.2.2.2
* DESCRIPTION: This procedure increments the send state
* variables SEND$STATE$B OR SEND$STATE$B when called to the
* the next modulo 8 sequence number to transmit. This
* procedure also performs the window function necessary
* for the LAP B protocol. Thawindow is currently 1.
* PASSED VARIABLES: CHANNEL, SEQ$NUM
* RETURNS: None
* GLOBAL VARIABLES USED: SEND$STATE$A, SEND$STATE$B
* GLOBAL VARIABLES CHANGED: SEND$STATE$A, SEND$STATE$B
* MODULES CALLED: INITACK, SEND$SEQ, PRINTI
* CALLING MODULES: SNO$RR,
* ALGORITHM: CASE OF CHANNEL
* CHANNEL A: IF SEND$STATE$A <= RCV$STATE$A + 1
* THEN INCREMENT SEND$STATE$A
* ELSE DO NOT INCREMENT
* END IF
* CHANNEL B: IF SEND$STATE$B <= RCV$STATE$B + 1
* THEN INCREMENT SEND$STATE$B
* ELSE DO NOT INCREMENT
* END IF
* END CASE OF CHANNEL
*
* AUTHOR: Mark Weber
* HISTORY: 1.0 Mark Weber -29 AUG 85-original PL/M version.
*
UP$DATE$SEND$STATE: PROCEDURE(CHANNEL, SEQ$NUM) PUBLIC;
*****/

```

SOURCE CODE FOR LAPBO.SRC MODULE TO NETX.25, 5 NOV 85

```

DECLARE (CHANNEL, SEQ$NUM) BYTE;
DECLARE TP$69(*) BYTE DATA(CR,LF,
    ERROR IN CHANNEL SELECTION FOR UP$DATA$SEND$STATE');
TP$69
DECLARE TP$69A(*) BYTE DATA(CR,LF,
    SEND$STATE$A HAS BEEN UPDATED: ');
TP$69A
DECLARE TP$69B(*) BYTE DATA(CR,LF,
    SEND$STATE$B HAS BEEN UPDATED: ');
TP$69B

IF ((CHANNEL = 1) OR (CHANNEL = 2)) THEN
DO CASE CHANNEL:
/* CASE 0 IS NULL */
DO:
IF (SEQ$NUM = (SEND$STATE$A + 1) MOD 8) THEN
CALL INITACK(CHANNEL + 2);
SEND$STATE$A = SEQ$NUM;
CALL SENDSEQ(@TP$69A, LENGTH(TP$69A));
CALL PRINTI(INT(SEND$STATE$A));
END; /* CASE 1 */
DO; /* CASE 2 */
IF (SEQ$NUM = (SEND$STATE$B + 1) MOD 8) THEN
CALL INITACK(CHANNEL + 2);
SEND$STATE$B = SEQ$NUM;
CALL SENDSEQ(@TP$69B, LENGTH(TP$69B));
CALL PRINTI(INT(SEND$STATE$B));
END; /* CASE 2 */
/* END CASE */
ELSE CALL SENDSEQ(@TP$69, LENGTH(TP$69));
END UP$DATA$SEND$STATE;

/*****
* DATE: 29 AUG 85
* VERSION: 1.0
* NAME: SND$CMDR
* MODULE NUMBER: 9.2.9
* DESCRIPTION: This procedure issues a CMDR FRAME to the
* transmit buffer of the UNID network when called. The
* procedure sends the control byte of the frame causing
* the error condition and sets bits within the information
* byte to reflect the conditions causing the error.
*
* PASSED VARIABLES: CHANNEL, PSBIT, CNTL$ERR, ERROR$STATUS
* RETURNS: None
* GLOBAL VARIABLES USED: TX01NE, TX02NE, NT01TX, NT02TX
* GLOBAL VARIABLES CHANGED: TX01NE, TX02NE, NT01TX, NT02TX
* MODULES CALLED: UP$DATA$SEND$STATE, LD$TAB$H$SKP, SENDSEQ
* CALLING MODULES: RCV$I$FRAME
* AUTHOR: Mark Weber
* HISTORY: 1.0 Mark Weber -29 AUG 85-original PL/M version.
*****/
SND$CMDR: PROCEDURE(CHANNEL, PSBIT, CNTL$ERROR, ERROR$STATUS) PUBLIC;

```

SOURCE CODE FOR LAPB0.SRC MODULE TO NETX.25, 5 NOV 85

```

DECLARE (CHANNEL, P$BIT, CNTL$ERROR, ERROR$STATUS) BYTE;
DECLARE CONTROL$BYTE BYTE;
DECLARE STATE$BYTE;
DECLARE TP$83(*) BYTE DATA(CR,LF,
    ERROR IN SND$CMDR CHANNEL NUMBER');
DECLARE TP$83A(*) BYTE DATA(CR,LF,
    ENTERING SND$CMDR, CHANNEL A');
DECLARE TP$83B(*) BYTE DATA(CR,LF,
    ENTERING SND$CMDR, CHANNEL B');

STATE = 0;
IF (P$BIT = 1) THEN
    CONTROL$BYTE = CMDR$CNTL OR 010H;
ELSE CONTROL$BYTE = CMDR$CNTL;
IF ((CHANNEL = 1) OR (CHANNEL = 2)) THEN
    DO CASE CHANNEL;
        /* CASE 0 IS NULL */
        /* CASE 1 */
        DO; CALL SENDSEQ(@TP$83A, LENGTH(TP$83A));
        SABM$MODE$A = TRUE;
        STATE = (SHL(SEND$STATES$A, 4) OR RCV$STATES$A);
        NT01TX(TX01NE + 0) = ASADD;
        NT01TX(TX01NE + 1) = CONTROL$BYTE;
        NT01TX(TX01NE + 2) = CNTL$ERROR;
        NT01TX(TX01NE + 3) = STATE;
        NT01TX(TX01NE + 4) = ERROR$STATUS;
        CALL LD$TAB$H$SKP(3, I$FRAME$SIZE);
        END; /* END CASE 1 */
        /* CASE 2 */
        DO; CALL SENDSEQ(@TP$83B, LENGTH(TP$83B));
        SABM$MODE$B = TRUE;
        STATE = (SHL(SEND$STATES$B, 4) OR RCV$STATES$B);
        NT02TX(TX02NE + 0) = BSADD;
        NT02TX(TX02NE + 1) = CONTROL$BYTE;
        NT02TX(TX02NE + 2) = CNTL$ERROR;
        NT02TX(TX02NE + 3) = STATE;
        NT02TX(TX02NE + 4) = ERROR$STATUS;
        CALL LD$TAB$H$SKP(4, I$FRAME$SIZE);
        END; /* END CASE 2 */
    END; /* END CASE */
ELSE CALL SENDSEQ(@TP$83, LENGTH(TP$83)); /* ERROR IN CHANNEL NUMBER */
END SND$CMDR;

/*****
* DATE: 29 AUG 85
* VERSION: 1.0
* NAME: RESET$QUE
* MODULE NUMBER: None
* DESCRIPTION: This procedure places the next to send
* pointer (OUTFRAMES$H$SNS) at the location of the I
*
*****/

```

SOURCE CODE FOR LAP80.SRC MODULE TO NETX.25, 5 NOV 85

```

* FRAME indicated by the sequence number. The procedure
* first searches for I FRAMES and then examines them for
* the correct sequence number. Once the correct sequence
* number is found, if it is found, the next to send
* pointer is reset. Transmission will begin at that point
* and every frame at the found sequence number or greater
* will be sent. A CMDR frame is sent if no sequence
* number match is found. This procedure is not used with
* the current single frame window protocol, but is
* included for demonstration purposes.
*
* PASSED VARIABLES:
*   CHANNEL, SEQ$NUM
*
* RETURNS:
*   None
*
* GLOBAL VARIABLES USED:
*   TX01NE, TX02NE
*
* GLOBAL VARIABLES CHANGED:
*   TX01NE, TX02NE
*
* MODULES CALLED:
*   None
*
* CALLING MODULES:
*   None
*
* AUTHOR: Mark Weber
*
* HISTORY: 1.0 Mark Weber -29 AUG 85-original PL/M version.
*****
RESET$QUE: PROCEDURE(CHANNEL, SEQ$NUM) PUBLIC;
*****

DECLARE (CHANNEL, SEQ$NUM) BYTE;
DECLARE INDEX INTEGER;
DECLARE FRAMES$CNTL BYTE;
DECLARE (SEQ$FOUND, SEQ$PURGED) BYTE;
DECLARE TP$B1(*) BYTE DATA(CR,LF,
'TP$B1 ERROR IN RESET$QUE CHANNEL NUMBER');
DECLARE TP$B1A(*) BYTE DATA(CR,LF,
'TP$B1A ENTERING RESET$QUE, CHANNEL A');
DECLARE TP$B1B(*) BYTE DATA(CR,LF,
'TP$B1B ENTERING RESET$QUE, CHANNEL B');

SEQ$FOUND = FALSE;
SEQ$PURGED = FALSE;
IF ((CHANNEL = 1) OR (CHANNEL = 2)) THEN
DO CASE CHANNEL:
; /* CASE 0 IS NULL */
DO; /* CASE 1 */
CALL SENDSEQ(@TP$B1A, LENGTH(TP$B1A));
INDEX = TX01NS;
DO WHILE NOT (SEQ$FOUND OR SEQ$PURGED);
FRAMES$CNTL = NT01TX(INDEX + 1);
IF ((FRAMES$CNTL AND 01H) = 0) THEN
IF ((FRAMES$CNTL AND 0E0H) = SHL(SEQ$NUM, 5)) THEN
DO;
TX01NS = INDEX;
SEQ$FOUND = TRUE;
END;
ELSE SEQ$FOUND = FALSE;
IF NOT SEQ$FOUND THEN

```

SOURCE CODE FOR LAPB0.SRC MODULE TO NETX.25, 5 NOV 85

```

DO; IF ((INDEX - I$FRAME$SIZE) >= 0) THEN
    INDEX = INDEX - I$FRAME$SIZE;
ELSE
    INDEX = FRAME$TABLE$SIZE - I$FRAME$SIZE;
    IF (INDEX = TX01NS) THEN
        SEQ$PURGED = TRUE;

END; /* END WHILE */
IF SEQ$PURGED THEN
    /* THE NEXT FRAME TRANSMITTED IS A CMDR FRAME */
    DO; CALL SND$CMDR(1,1, FRAME$CNTL, 0BH);
        TX01NS = TX01NE;
        CALL LD$TAB$H$SKP(3, IFRAME$SIZE);
    END;
    /* END CASE 1 */
    /* CASE 2 */
    DO; CALL SENDSEQ(@TP$B1B, LENGTH(TP$B1B));
        INDEX = TX02NS;
        DO WHILE NOT (SEQ$FOUND OR SEQ$PURGED);
            FRAME$CNTL = NT02TX(INDEX + 1);
            IF ((FRAME$CNTL AND 01H) = 0) THEN
                IF ((FRAME$CNTL AND 0EOH) = SHL(SEQ$NUM, 5)) THEN
                    DO; TX02NS = INDEX;
                        SEQ$FOUND = TRUE;
                    END;
                ELSE SEQ$FOUND = FALSE;
            IF NOT SEQ$FOUND THEN
                DO; IF ((INDEX - I$FRAME$SIZE) >= 0) THEN
                    INDEX = INDEX - I$FRAME$SIZE;
                ELSE
                    INDEX = FRAME$TABLE$SIZE - I$FRAME$SIZE;
                    IF (INDEX = TX02NS) THEN
                        SEQ$PURGED = TRUE;
                END;
            /* END WHILE */
            IF SEQ$PURGED THEN
                DO; /* THE NEXT FRAME TRANSMITTED IS A CMDR FRAME */
                    CALL SND$CMDR(2,1, FRAME$CNTL, 0BH);
                    TX02NS = TX02NE;
                    CALL LD$TAB$H$SKP(3, IFRAME$SIZE);
                END;
            /* END CASE 2 */
            /* END CASE */
            ELSE CALL SENDSEQ(@TP$B1, LENGTH(TP$B1));
        END RESET$QUE;
    /* ***** */

```


SOURCE CODE FOR LAP80.SRC MODULE TO NETX.25, 5 NOV 85

```

* DATE: 28 AUG 85
* VERSION: 1.0
* NAME: SNDSOM
* MODULE NUMBER: 8.4
* DESCRIPTION: This procedure sends a DM response
*              U frame in response to any frame other than a SABM
*              command while not in the SABM mode.
* PASSED VARIABLES: CHANNEL
* RETURNS: None
* GLOBAL VARIABLES USED: NTOINE, NT02NE
* GLOBAL VARIABLES CHANGED: NTOINE, NT02NE
* MODULES CALLED: LD$TAB$H$SKP, SENDSEQ
* CALLING MODULES: START$DM$MODE,
* AUTHOR: Mark Weber
* HISTORY: 1.0 Mark Weber
*****
SNDSOM: PROCEDURE(CHANNEL, P$BIT) PUBLIC;
*****
DECLARE (CHANNEL, P$BIT) BYTE;
DECLARE CONTROL$BYTE BYTE;
DECLARE TP88(*) BYTE DATA(CR,LF);
TP88 SNDSOM HAS ERROR IN CHANNEL SELECTION ',CR,LF);
DECLARE TP$88A(*) BYTE DATA(CR,LF, CHANNEL A');
TP$88A ENTERING SNDSOM;
DECLARE TP$88B(*) BYTE DATA(CR,LF, CHANNEL B');
TP$88B ENTERING SNDSOM, CHANNEL B');

IF ((CHANNEL = 1) OR (CHANNEL = 2)) THEN
DO CASE CHANNEL:
; /* CASE 0 IS NULL */
DO; /* CASE 1 FOR CHANNEL A
CALL SENDSEQ(@TP$88A, LENGTH(TP$88A));
NT01TX(TX01NE + 0) = B$ADD; /* DM IS A RESPONSE */
IF P$BIT = FALSE THEN
CONTROL$BYTE = DM$CNTL;
ELSE
CONTROL$BYTE = DM$CNTL OR 10H;
NT01TX(TX01NE + 1) = CONTROL$BYTE;
CALL LD$TAB$H$SKP(3, 1$FRAME$SIZE);
END; /* CASE 1 */
*/

DO; /* CASE 2 FOR CHANNEL B
CALL SENDSEQ(@TP$88B, LENGTH(TP$88B));
NT02TX(TX02NE + 0) = A$ADD; /* DM IS A RESPONSE */
IF P$BIT = FALSE THEN
CONTROL$BYTE = DM$CNTL;
ELSE
CONTROL$BYTE = DM$CNTL OR 10H;
NT02TX(TX02NE + 1) = CONTROL$BYTE;
CALL LD$TAB$H$SKP(4, 1$FRAME$SIZE);
*/

```

SOURCE CODE FOR LAPBO.SRC MODULE TO NETX.25, 5 NOV 85

```

END; /* CASE 2 */
END; /* END CASE */

ELSE CALL SENDSEQ(@TP88, LENGTH(TP$88));
END SND$DM;

/*****
* DATE: 28 AUG 85
* VERSION: 1.0
* NAME: SND$SABM
* MODULE NUMBER: 8.5
* DESCRIPTION: This procedure sends a SABM command
* U frame for transmission to the channel indicated by
* the passed parameter.
* CHANNEL
* PASSED VARIABLES:
* RETURNS: None
* GLOBAL VARIABLES USED: NTO1NE, NTO2NE, TXO1NE, TXO2NE
* GLOBAL VARIABLES CHANGED: NTO1NE, NTO2NE, TXO1NE, TXO2NE
* MODULES CALLED: LD$TAB$H$SKP, STOS$US$CMD, SENDSEQ
* CALLING MODULES: START$DM$MODE,
* AUTHOR: Mark Weber
* HISTORY: 1.0 Mark Weber
*****/
SND$SABM: PROCEDURE(CHANNEL, P$BIT) PUBLIC;

DECLARE (CHANNEL, P$BIT) BYTE;
DECLARE CONTROL$BYTE BYTE;
DECLARE TP60(*) BYTE DATA(CR,LF,
'TP60
SND$SABM DOES NOT HAVE TABLE SPECIFIED',CR,LF);
DECLARE TP$60A(*) BYTE DATA(CR,LF,
'TP$60A
ENTERING SND$SABM, CHANNEL A');
DECLARE TP$60B(*) BYTE DATA(CR,LF,
'TP$60B
ENTERING SND$SABM, CHANNEL B');

IF (CHANNEL = 1 OR CHANNEL = 2) THEN
DO CASE CHANNEL;
/* CASE 0 IS NULL */
DO: /* CASE 1 FOR CHANNEL A
CALL SENDSEQ(@TP$60A, LENGTH(TP$60A));
NTO1TX(TXO1NE + 0) = A$ADD;
IF P$BIT = FALSE THEN
CONTROL$BYTE = SABM$CNTL;
ELSE
CONTROL$BYTE = SABM$CNTL OR 10H;
NTO1TX(TXO1NE + 1) = CONTROL$BYTE;
CALL STOS$US$CMD(1, CONTROL$BYTE);
CALL LD$TAB$H$SKP(3, I$FRAME$SIZE);
END; /* CASE 1
*/

```

SOURCE CODE FOR LAPBO.SRC MODULE TO NETX.25, 5 NOV 85

```

DO:      /* CASE 2 FOR CHANNEL B */
CALL SEND$SEQ(@TP$60B, LENGTH(TP$60B));
NT02TX(TX02NE + 0) = B$ADD;
IF P$BIT = FALSE THEN
    CONTROL$BYTE = SABM$CNTL;
ELSE
    CONTROL$BYTE = SABM$CNTL OR 10H;
NT02TX(TX02NE + 1) = CONTROL$BYTE;
CALL ST0$USCMD(2, CONTROL$BYTE);
CALL LD$TAB$H$KP(4, I$FRAME$SIZE);
END;      /* CASE 2 */
*/

END;      /* END CASE */
*/

ELSE CALL SEND$SEQ(@TP60, LENGTH(TP$60));
END SND$SABM;

/*****
* DATE: 28 AUG 85
* VERSION: 1.0
* NAME: RCV$IN$SEQ
* MODULE NUMBER: 9.2.2.1
* DESCRIPTION: This function returns a TRUE value if
the sequence number in the received I$FRAME is the one
expected, otherwise FALSE is returned.
* PASSED VARIABLES: CHANNEL, SEQ$NUM
* RETURNS: SEQUENCE
* GLOBAL VARIABLES USED: RCV$STATE$, RCV$STATE$B
* GLOBAL VARIABLES CHANGED: All the above
* MODULES CALLED: PRINT1, SEND$SEQ
* CALLING MODULES: RCV$I$FRAME, RCV$RR, RCV$RNR, RCV$REJ
* AUTHOR: Mark Weber
* HISTORY: 1.0 Mark Weber -28 AUG 85-original PL/M version
*****
RCV$IN$SEQ: PROCEDURE(CHANNEL, SEQ$NUM) BYTE PUBLIC;

DECLARE (CHANNEL, SEQ$NUM) BYTE;
DECLARE SEQUENCE BYTE;
DECLARE TP$49(*) BYTE DATA(CR,LF,
'TP$49 RCV$SEQ$A IS IN ERROR: ');
DECLARE TP$50(*) BYTE DATA(CR,LF,
'TP$50 RCV$SEQ$B IS IN ERROR: ');
DECLARE TP$51(*) BYTE DATA(CR,LF,
'TP$51 RCV$SEQ HAS CHANNEL ERROR');

SEQUENCE = FALSE;
IF ((CHANNEL = 1) OR (CHANNEL = 2)) THEN
DO CASE CHANNEL;
: /* CASE 0 IS NULL */
DO: /* CASE 1 */

```

SOURCE CODE FOR LAP80.SRC MODULE TO NETX.25, 5 NOV 85

```

IF (RCV$STATE$A = SEQ$NUM) THEN
    SEQUENCE = TRUE;
ELSE
    DO;
        CALL SENDSEQ(@TP$49, LENGTH(TP$49));
        CALL PRINTI(INT(RCV$STATE$A));
    END;
END; /* CASE 1 */

DO; /* CASE 2 */
    IF (RCV$STATE$B = SEQ$NUM) THEN
        SEQUENCE = TRUE;
    ELSE
        DO;
            CALL SENDSEQ(@TP$50, LENGTH(TP$50));
            CALL PRINTI(INT(RCV$STATE$B));
        END;
END; /* CASE 2 */

END; /* END CASE */
ELSE CALL SENDSEQ(@TP$51, LENGTH(TP$51));

RETURN SEQUENCE;
END RCV$IN$SEQ;

/*****
* DATE: 28 AUG 85
* VERSION: 1.0
* NAME: SND$RR
* MODULE NUMBER: 9.2.2.6
* DESCRIPTION: This procedure sends a RR FRAME when
*               invoked by the calling routine.
* PASSED VARIABLES: CHANNEL, P$BIT
* RETURNS: None
* GLOBAL VARIABLES USED: NT01TX, TX01NE, TX02NE
* GLOBAL VARIABLES CHANGED: all the above
* MODULES CALLED: LOSTAB$H$SKP, SENDSEQ
* CALLING MODULES: RCV$1$FRAME, READ$LINE
* AUTHOR: Mark Weber
* HISTORY: 1.0 Mark Weber -28 AUG 85-original PL/M version
*****
SND$RR: PROCEDURE(CHANNEL, P$BIT, ADDR) PUBLIC REENTRANT;

    DECLARE (CHANNEL, P$BIT, ADDR) BYTE;
    DECLARE CONTROL$BYTE BYTE;
    DECLARE TP$70(*) BYTE DATA(CR,LF,
    'TP$70
    ERROR IN CHANNEL SELECTION FOR SND$RR');
    DECLARE TP$70A(*) BYTE DATA(CR,LF,
    'TP$70A
    ENTERING SND$RR, CHANNEL A');
    DECLARE TP$70B(*) BYTE DATA(CR,LF,
    'TP$70B
    ENTERING SND$RR, CHANNEL B');

```

SOURCE CODE FOR LAPBO.SRC MODULE TO NETX.25, 5 NOV 85

```

IF ((CHANNEL = 1) OR (CHANNEL = 2)) THEN
DO CASE CHANNEL:
/* CASE 0 IS NULL */
DO:/* CASE 1 */
CALL SENDSEQ(@TP$70A, LENGTH(TP$70A));
IF (P$BIT = 1) THEN
CONTROL$BYTE = (RR$CNTL OR (010H OR SHL(RCV$STATE$A, 5)));
ELSE CONTROL$BYTE = (RR$CNTL OR SHL(RCV$STATE$A, 5));
NT01TX(TX01NE + 0) = ADDR; /* RR FRAME IS A RESPONSE */
NT01TX(TX01NE + 1) = CONTROL$BYTE;
CALL LD$TAB$H$SKP(3, 1$FRAME$SIZE);
END; /* END CASE 1 */
DO:/* CASE 2 */
CALL SENDSEQ(@TP$70B, LENGTH(TP$70B));
IF (P$BIT = 1) THEN
CONTROL$BYTE = (RR$CNTL OR (010H OR SHL(RCV$STATE$B, 5)));
ELSE CONTROL$BYTE = (RR$CNTL OR SHL(RCV$STATE$B, 5));
NT02TX(TX02NE + 0) = ADDR; /* RR FRAME IS A RESPONSE */
NT02TX(TX02NE + 1) = CONTROL$BYTE;
CALL LD$TAB$H$SKP(4, 1$FRAME$SIZE);
END; /* END CASE 2 */
END; /* END CASE */
ELSE CALL SENDSEQ(@TP$70, LENGTH(TP$70));
END SND$RR;

```

```

/*****
* DATE: 29 AUG 85
* VERSION: 1.0
* NAME: SND$REJ
* MODULE NUMBER: 9.2.2.9
* DESCRIPTION: This procedure issues a REJ FRAME to the
* transmit buffer of the UNID network when called.
* PASSED VARIABLES: CHANNEL, P$BIT
* RETURNS: None
* GLOBAL VARIABLES USED: NT01TX, NT02TX, TX01NE, TX02NE
* GLOBAL VARIABLES CHANGED: NT01TX, NT02TX
* MODULES CALLED: LD$TAB$H$SKP, SENDSEQ
* CALLING MODULES: RCV$I$FRAME, READ$LINE
* AUTHOR: Mark Weber
* HISTORY: 1.0 Mark Weber -29 AUG 85-original PL/M version.
*****/
SND$REJ: PROCEDURE(CHANNEL, P$BIT, ADDR) PUBLIC;

DECLARE (CHANNEL, P$BIT, ADDR) BYTE;
DECLARE CONTROL$BYTE BYTE;
DECLARE TP$74(*) BYTE DATA(CR,LF,
'TP$74
DECLARE TP$74A(*) BYTE DATA(CR,LF,
'TP$74A
ENTERING SND$REJ, CHANNEL A');

```

SOURCE CODE FOR LAP80.SRC MODULE TO NETX.25, 5 NOV 85

```

DECLARE TP$74B(*) BYTE DATA(CR,LF,
'TP$74B' ENTERING SND$REJ, CHANNEL B');

IF ((CHANNEL = 1) OR (CHANNEL = 2)) THEN
DO CASE CHANNEL;
: /* CASE 0 IS NULL */
DO; /* CASE 1 */
CALL SENDSEQ(@TP$74A, LENGTH(TP$74A));
IF (P$BIT = 1) THEN
CONTROL$BYTE = (REJ$CNTL OR (Q10H OR SHL(RCV$STATES$A, 5)));
ELSE CONTROL$BYTE = (REJ$CNTL OR SHL(RCV$STATES$A, 5));
NT01TX(TX01NE + 0) = ADDR; /* REJ IS A COMMAND */
NT01TX(TX01NE + 1) = CONTROL$BYTE;
CALL LD$TAB$H$SKP(3, I$FRAME$SIZE);
END; /* END CASE 1 */
DO; /* CASE 2 */
CALL SENDSEQ(@TP$74B, LENGTH(TP$74B));
IF (P$BIT = 1) THEN
CONTROL$BYTE = (REJ$CNTL OR (Q10H OR SHL(RCV$STATES$B, 5)));
ELSE CONTROL$BYTE = (REJ$CNTL OR SHL(RCV$STATES$B, 5));
NT02TX(TX02NE + 0) = ADDR; /* REJ IS A COMMAND */
NT02TX(TX02NE + 1) = CONTROL$BYTE;
CALL LD$TAB$H$SKP(4, I$FRAME$SIZE);
END; /* END CASE 2 */
/* END CASE */
ELSE CALL SENDSEQ(@TP$74, LENGTH(TP$74)); /* ERROR IN CHANNEL NUMBER */
END SND$REJ;

/*****
* DATE: 29 AUG 85
* VERSION: 1.0
* NAME: SND$RNR
* MODULE NUMBER: 7.8
* DESCRIPTION: This procedure issues a RNR FRAME to the
* transmit buffer of the UNID network when called. The
* procedure sets the global variable RNR$MODE to TRUE.
* This suspends the ROUTE$IN procedure's ability to
* any more I$FRAMES. The condition is cleared when a
* a RR or REJ frame is received by ROUTE$IN.
* PASSED VARIABLES: CHANNEL, P$BIT
* RETURNS: None
* GLOBAL VARIABLES USED: NT01TX, NT01TX, TX01NE, TX02NE,
* RNR$MODE$A, RNR$MODE$B
* GLOBAL VARIABLES CHANGED:
* MODULES CALLED:
* CALLING MODULES: LD$STASH$SKP
* AUTHCR: Mark Weber RCV$I$FRAME, READ$LINE
* HISTORY: 1.0 Mark Weber -29 AUG 85-original PL/M version.
* *****/
SND$RNR: PROCEDURE(CHANNEL, P$BIT, ADDR) PUBLIC;

```

SOURCE CODE FOR LAPBO.SRC MODULE TO NETX.25, 5 NOV 85

```

DECLARE (CHANNEL, P$BIT, ADDR) BYTE;
DECLARE CONTROL$BYTE BYTE;
DECLARE TP$71(*) BYTE DATA(CR,LF,
    'TP$71' ERROR IN SDRNR CHANNEL NUMBER');
DECLARE TP$71A(*) BYTE DATA(CR,LF,
    'TP$71A' ENTERING SDRNR, CHANNEL A');
DECLARE TP$71B(*) BYTE DATA(CR,LF,
    'TP$71B' ENTERING SDRNR, CHANNEL B');

IF ((CHANNEL = 1) OR (CHANNEL = 2)) THEN
DO CASE CHANNEL:
/* CASE 0 IS NULL */
DO: /* CASE 1 */
CALL SENDSEQ(@TP$71A, LENGTH(TP$71A));
IF (P$BIT = 1) THEN
CONTROL$BYTE = (RNR$CNTL OR (010H OR SHL(RCV$STATE$A, 5)));
ELSE CONTROL$BYTE = (RNR$CNTL OR SHL(RCV$STATE$A, 5));
RNR$MODE$A = TRUE;
NT01TX(TX01NE + 0) = ADDR; /* RNR IS A COMMAND */
NT01TX(TX01NE + 1) = CONTROL$BYTE;
CALL LD$TAB$H$SKP(3, I$FRAME$SIZE);
END; /* END CASE 1 */
DO: /* CASE 2 */
CALL SENDSEQ(@TP$71B, LENGTH(TP$71B));
IF (P$BIT = 1) THEN
CONTROL$BYTE = (RNR$CNTL OR (010H OR SHL(RCV$STATE$B, 5)));
ELSE CONTROL$BYTE = (RNR$CNTL OR SHL(RCV$STATE$A, 5));
RNR$MODE$B = TRUE;
NT02TX(TX02NE + 0) = ADDR; /* RNR IS A COMMAND */
NT02TX(TX02NE + 1) = CONTROL$BYTE;
CALL LD$TAB$H$SKP(4, I$FRAME$SIZE);
END; /* END CASE 2 */
END; /* END CASE */
ELSE CALL SENDSEQ(@TP$71, LENGTH(TP$71)); /* ERROR IN CHANNEL NUMBER */
END SDRNR;

/*****
* DATE: 29 AUG 85
* VERSION: 1.0
* NAME: SDR$UA
* MODULE NUMBER: 8.2.2
* DESCRIPTION: This procedure issues a UA FRAME to the
* transmit buffer of the UNID network when called. The
* procedure sets the global variable RNR$MODE to TRUE.
* This suspends the ROUTESIN procedure's ability to
* any more I$FRAMES. The condition is cleared when a
* a RR or REJ frame is received by ROUTESIN.
* CHANNEL, P$BIT
* PASSED VARIABLES: None
* RETURNS: NT01TX, TX01NE, TX02NE
* GLOBAL VARIABLES USED:
*****/

```

SOURCE CODE FOR LAPBO.SRC MODULE TO NETX.25, 5 NOV 85

```

* GLOBAL VARIABLES CHANGED:  NT01TX, NT02TX, TX01NE, TX02NE
* MODULES CALLED:           LD$TAB$H$SKP
* CALLING MODULES:          RCV$1$FRAME, READ$LINE
* AUTHOR: Mark Weber
* HISTORY: 1.0 Mark Weber -29 AUG 85-original PL/M version.
*****
SND$UA: PROCEDURE(CHANNEL, P$BIT) PUBLIC REENTRANT;

```

```

DECLARE (CHANNEL, P$BIT) BYTE;

```

```

DECLARE CONTROL$BYTE BYTE;

```

```

DECLARE TP$72(*) BYTE DATA(CR,LF,
'TP$72 ERROR IN SND$UA CHANNEL NUMBER');

```

```

DECLARE TP$72A(*) BYTE DATA(CR,LF,
'TP$72A ENTERING SND$UA, CHANNEL A');

```

```

DECLARE TP$72B(*) BYTE DATA(CR,LF,
'TP$72B ENTERING SND$UA, CHANNEL B');

```

```

IF (P$BIT = 1) THEN

```

```

    CONTROL$BYTE = (UASCNTL OR 010H);

```

```

ELSE CONTROL$BYTE = UASCNTL;

```

```

IF ((CHANNEL = 1) OR (CHANNEL = 2)) THEN

```

```

    DO CASE CHANNEL;

```

```

        ; /* CASE 0 IS NULL */

```

```

        DO; /* CASE 1 */

```

```

            CALL SENDSEQ(@TP$72A, LENGTH(TP$72A));

```

```

            NT01TX(TX01NE + 0) = BSADD; /* UA IS A RESPONSE */

```

```

            NT01TX(TX01NE + 1) = CONTROL$BYTE;

```

```

            CALL LD$TAB$H$SKP(3, I$FRAME$SIZE);

```

```

        END; /* END CASE 1 */

```

```

        DO; /* CASE 2 */

```

```

            CALL SENDSEQ(@TP$72B, LENGTH(TP$72B));

```

```

            NT02TX(TX02NE + 0) = ASADD; /* US IS A RESPONSE */

```

```

            NT02TX(TX02NE + 1) = CONTROL$BYTE;

```

```

            CALL LD$TAB$H$SKP(4, I$FRAME$SIZE);

```

```

        END; /* END CASE 2 */

```

```

        ; /* END CASE */

```

```

    ELSE CALL SENDSEQ(@TP$72, LENGTH(TP$72)); /* ERROR IN CHANNEL NUMBER */

```

```

END SND$UA;

```

```

/*****
* DATE: 29 AUG 85
* VERSION: 1.0
* NAME: SND$DISC
* MODULE NUMBER: 7.11
* DESCRIPTION: This procedure issues a DISC FRAME to the
* transmit buffer of the UNID network when called. The
* procedure sets the global variable SABM$MODE to FALSE.
* This suspends the START$DATA$XFER procedure and control
* returns to the main program where the DM mode is again
*
*****/

```


SOURCE CODE FOR LAPB0.SRC MODULE TO NETX.25, 5 NOV 85

```

*
*   entered.
*   CHANNEL, PSBIT
*
* PASSED VARIABLES:
* RETURNS:
* GLOBAL VARIABLES USED: NT01TX, NT02TX, TX01NE, TX02NE
* GLOBAL VARIABLES CHANGED: NT01TX, NT02TX, TX01NE, TX02NE
* MODULES CALLED: LD$TAB$H$SKP
* CALLING MODULES: RCV$I$FRAME, READ$LINE
*
* AUTHOR: Mark Weber
* HISTORY: 1.0 Mark Weber -29 AUG 85-original PL/M version.
* *****
*
* SNO$DISC: PROCEDURE(CHANNEL, PSBIT) PUBLIC;
*
*   DECLARE (CHANNEL, PSBIT) BYTE;
*   DECLARE CONTROL$BYTE BYTE;
*   DECLARE TP$73(*) BYTE DATA(CR,LF,
*   'TP$73 ERROR IN DISC$UA CHANNEL NUMBER');
*   DECLARE TP$73A(*) BYTE DATA(CR,LF,
*   'TP$73A ENTERING SNO$DISC, CHANNEL A');
*   DECLARE TP$73B(*) BYTE DATA(CR,LF,
*   'TP$73B ENTERING SNO$DISC, CHANNEL B');
*
* IF (PSBIT = 1) THEN
*   CONTROL$BYTE = (DISC$CNTL OR 010H);
* ELSE CONTROL$BYTE = DISC$CNTL;
* IF ((CHANNEL = 1) OR (CHANNEL = 2)) THEN
*   DO CASE CHANNEL:
*   ; /* CASE 0 IS NULL */
*   DO; /* CASE 1 */
*   CALL SENDSEQ(@TP$73A, LENGTH(TP$73A));
*   NT01TX(TX01NE + 0) = ASADD; /* DISC IS A COMMAND */
*   NT01TX(TX01NE + 1) = CONTROL$BYTE;
*   CALL STOSU$CMD$(1, DISC$CNTL);
*   CALL LD$TAB$H$SKP(3, I$FRAME$SIZE);
*   END; /* END CASE 1 */
*   DO; /* CASE 2 */
*   CALL SENDSEQ(@TP$73B, LENGTH(TP$73B));
*   NT02TX(TX02NE + 0) = BSADD; /* DISC IS A COMMAND */
*   NT02TX(TX02NE + 1) = CONTROL$BYTE;
*   CALL STOSU$CMD$(2, DISC$CNTL);
*   CALL LD$TAB$H$SKP(4, I$FRAME$SIZE);
*   END; /* END CASE 2 */
*   END; /* END CASE */
* ELSE CALL SENDSEQ(@TP$73, LENGTH(TP$73)); /* ERROR IN CHANNEL NUMBER */
* END SNO$DISC;
*
* *****
* * DATE: 28 AUG 85
* * VERSION: 1.0
* * NAME: RCV$I$FRAME
*

```

SOURCE CODE FOR LAPBO.SRC MODULE TO NETX.25, 5 NOV 85

```

* MODULE NUMBER: 9.2.2
* DESCRIPTION: This procedure processes all I frames
* received by the network side of UNID II. Frames are
* sent to the appropriate transmit table and an acknow-
* ledgement RR frame is sent if no I frame is in the
* transmit buffer.
* PASSED VARIABLES: CHANNEL, P$BIT
* RETURNS: None
* GLOBAL VARIABLES USED: SEND$STATE$, RCV$STATE$, SEND$STATE$,
* RCV$STATE$, NT01TX, NT02TX, LCNTTB, TX01NS,
* TX02NS
* GLOBAL VARIABLES CHANGED: SEND$STATE$, RCV$STATE$, SEND$STATE$,
* RCV$STATE$, NT01TX, NT02TX, TX01NS, TX02NS
* MODULES CALLED: IN$SEQ
* CALLING MODULES: ROUTE$IN
* AUTHOR: Mark Weber
* HISTORY: 1.0 Mark Weber -28 AUG 85-original PL/M version
*****
RCV$I$FRAME: PROCEDURE(CHANNEL, P$BIT) PUBLIC;

DECLARE(CHANNEL, P$BIT) BYTE;
DECLARE (RCV$SEQNUM, SND$SEQNUM) BYTE;
DECLARE TP$6(*) BYTE(CR,LF);
TP$6 FRAME IS NT01RX TO NT02TX TRANSFER';
DECLARE TP$7(*) BYTE DATA(CR,LF);
TP$7 DATA IS DESTINED FOR THIS UNID';
DECLARE TP$15(*) BYTE DATA(CR,LF);
TP$15 FRAME IS NT02RX TO NT01TX TRANSFER';
DECLARE TP$16(*) BYTE DATA(CR,LF);
TP$16 I FRAME SEND TO THE NTLC TABLE';
DECLARE TP$67(*) BYTE DATA(CR,LF);
TP$67 Entering module RCV$I$FRAME';
DECLARE TP$68(*) BYTE DATA(CR,LF);
TP$68 ERROR IN I$FRAME CHANNEL CALL';
DECLARE TP$68A(*) BYTE DATA(CR,LF);
TP$68A ERROR IN I$FRAME SEQUENCE NUMBER';

IF (CHANNEL = 1 OR CHANNEL = 2) THEN
DO CASE CHANNEL;
/* CASE 0 IS NULL AND ERROR */
DO;
/* CASE 1
SND$SEQNUM = (SHR(NT01RX(NT01NS + 1), 1) AND 07);
RCV$SEQNUM = (SHR(NT01RX(NT01NS + 1), 5));
IF (NOT RNR$MODE$A) THEN
DO;
IF (RCV$INSEQ( 1, SND$SEQNUM)) THEN
DO;
RCV$STATE$ = (SND$SEQNUM + 1) MOD 8;
CALL UPDATE$SEND$STATE(1, RCV$SEQNUM);
IF (P$BIT = 1) THEN
/* SEND ACK BACK TO SENDER*/

```

```

DO; IF (NT01TX(TX01NS + 1) AND I$CNTL) = 0
    AND FRAME$PRESENT(1) THEN
    NT01TX(TX01NS + 1) = (SHL(RCV$STATE$A, 5) OR
    (NT01TX(TX01NS + 1) AND 01FH));
    ELSE CALL SND$RR(1,1,B$ADD);
END;
DESTINATION = DET$DEST$ONE;
IF (DESTINATION = 'NN') THEN /* FRAME BACK TO NETWORK */
DO; CALL SENDSEQ(@TP$6, LENGTH(TP$6));
    this kills the I frames for the simulation
    this code must be included for operation software
    CALL MOV(B,@NT01RX(NT01NS),
    @NT02TX(TX02NE), I$FRAME$SIZE);
    NT02TX(TX02NE + 1) = (SHL(RCV$STATE$B, 5) OR 010H)
    OR SHL(SENDS$STATE$B, 1);
    CALL LD$TAB$H$SKP(4,I$FRAME$SIZE);
    END;
ELSE /* DESTINATION = NL */ /* FRAME TO LOCAL TABLES */
DO; CALL SENDSEQ(@TP$7, LENGTH(TP$7));
    CALL MOV(B,@NT01RX(NT01NS + 2), @NTLCTB(NTLCLNE),
    PACKET$SIZE);
    CALL LD$TAB$H$SKP(5,PACKET$SIZE);
    END;
END;
ELSE /* MESSAGE WAS OUT OF SEQUENCE, SEND REJ FRAME */
DO; CALL SENDSEQ(@TP$68A, LENGTH(TP$68A));
    CALL SND$REJ(1,1,B$ADD);
    END;
END;
CALL SRVC$TAB$H$SKP(1, I$FRAME$SIZE);
END; /* CASE 1 */
/* CASE 2 */
SND$SEQ$NUM = (SHR(NT02RX(NT02NS + 1), 1) AND 07);
RCV$SEQ$NUM = (SHR(NT02RX(NT02NS + 1), 5));
IF (NOT RNR$MODE$B) THEN
DO;
    IF (RCV$INSEQ( 2, SND$SEQ$NUM)) THEN
    DO; RCV$STATE$B = (SND$SEQ$NUM + 1) MOD 8;
        CALL UPDATES$END$STATE(2, RCV$SEQ$NUM);
        IF (P$BIT = 1) THEN /* SEND ACK BACK TO SENDER */
        DO; IF ((NT02TX(TX02NS + 1) AND I$CNTL) = 0)
            AND FRAME$PRESENT(2) THEN
            NT01TX(TX01NS + 1) = (SHL(RCV$STATE$B, 5) OR

```

SOURCE CODE FOR LAPBO.SRC MODULE TO NETX.25, 5 NOV 85

```

      (NT02TX(TX02NS + 1),A$ADD));
    ELSE CALL SND$RRR(2,1,A$ADD);
  END;
  DESTINATION = DET$DEST$TWO;
  IF (DESTINATION = 'NN') THEN /* FRAME BACK TO NETWORK*/
DO;
  CALL SENDSEQ(@TP$15, LENGTH(TP$15));
  this kills the I frames for the simulation
  this code must be included for operation software ***
  CALL MOVV(@NT02RX(NT02NS),
  @NT01TX(TX01NE), I$FRAME$SIZE);
  NT01TX(TX01NE + 1) = (SHL(RCV$STATESA, 5) OR 010H)
  OR SHL(SEND$STATESA, 1); ***
  CALL LD$TAB$H$SKP(3,I$FRAME$SIZE);
END;
ELSE /* DESTINATION = NL */ /* FRAME TO LOCAL TABLES */
DO;
  CALL SENDSEQ(@TP$16, LENGTH(TP$16));
  CALL MOVV(@NT02RX(NT02NS + 2), @NTLCB(NTLCNE),
  PACKET$SIZE);
  CALL LD$TAB$H$SKP(5,PACKET$SIZE);
END;
END;
ELSE
DO; /* MESSAGE WAS OUT OF SEQUENCE, SEND REJ FRAME */
  CALL SENDSEQ(@TP$68A, LENGTH(TP$68A));
  CALL SND$REJ(2,1,A$ADD);
END;
END;
END;
CALL SRVC$TAB$H$SKP(2, I$FRAME$SIZE);
/* CASE 2
/* END CASE
ELSE CALL SENDSEQ(@TP$68, LENGTH(TP$68)); /* ERROR IN CHANNEL CALL */
END RCV$I$FRAME;

/*****
* DATE: 28 AUG 85
* VERSION: 1.0
* NAME: RCV$SABM
* MODULE NUMBER: 8.2
* DESCRIPTION:
  This procedure is invoked when a SABM
  frame is received. When a SABM frame is found, a UA
  frame is sent to the source and the SABM mode is entered.
  The receive table pointer to the next frame to process
  is then bumped ahead by one I FRAME size. Later ver-
  may make use of the variable frame adjustment to place
  more then ten frames in the receive table.
* PASSED VARIABLES:
* RETURNS:
* GLOBAL VARIABLES USED: NONE, NT02NE, TX01NE, TX02NE,
****

```

```

SOURCE CODE FOR LAP80.SRC MODULE TO NETX.25, 5 NOV 85

*
* NTOINS, NT02NS, TX01NE, TX02NS,
* SABM$MODE$A, SABM$MODF$B, RNR$MODE$A,
* RNR$MODE$B
*
* GLOBAL VARIABLES CHANGED: All globals used
*
* MODULES CALLED: SNDSUA
*
* CALLING MODULES: START$DM$MODE, ROUTE$IN
*
* AUTHOR: Mark Weber
*
* HISTORY: 1.0 Mark Weber -30 AUG 85- original translated PL/M
*
* ***** version *****
*
* RCV$SABM: PROCEDURE(CHANNEL, P$BIT)PUBLIC;
*
*
* DECLARE (CHANNEL, P$BIT) BYTE;
* DECLARE TP$61(*) BYTE DATA(CR,LF);
*
* TP$61 P BIT WAS NOT SET WHEN SABM WAS RECEIVED',CR,LF);
*
* DECLARE TP$61A(*) BYTE DATA(CR,LF);
*
* TP$61A ENTERING RCV$SABM, CHANNEL A');
*
* DECLARE TP$61B(*) BYTE DATA(CR,LF);
*
* TP$61B ENTERING RCV$SABM, CHANNEL B');
*
* IF (CHANNEL = 1 OR CHANNEL = 2) THEN
* DO CASE CHANNEL;
* : /*CASE 0 IS NULL */
* DO: /*CASE 1 IS FOR CHANNEL A */
* IF P$BIT = 1 THEN
* DO;
* CALL SNDSUA(1,1);
* SABM$MODE$A = TRUE;
* RNR$MODE$A = FALSE;
* SEND$STATE$A = 0;
* RCV$STATE$A = 0;
* TX01NS, TX01NE = 0;
* NT01NE, NT01NS = 0;
*
* END;
* ELSE CALL SENDSEQ(@TP$61, LENGTH(TP$61));
* END; /*CASE 1 */
*
* DO: /*CASE 2 IS FOR CHANNEL B */
* CALL SENDSEQ(@TP$61B, LENGTH(TP$61B));
* IF P$BIT = 1 THEN
* DO;
* CALL SNDSUA(2,1);
* SABM$MODE$B = TRUE;
* RNR$MODE$B = FALSE;
* SEND$STATE$B = 0;
* RCV$STATE$B = 0;
* TX02NS, TX02NE = 0;
* NT02NE, NT02NS = 0;
*
* END;

```

SOURCE CODE FOR LAPB0.SRC MODULE TO NETX.25, 5 NOV 85

```

        ELSE CALL SENDSEQ(@TP$61, LENGTH(TP$61));
        END; /*CASE 2 */
    END; /* END CASE */
END RCV$SABM;

```

```

/*****
* DATE: 28 AUG 85
* VERSION: 1.0
* NAME: RCV$DM
* MODULE NUMBER: 9.2.10
* DESCRIPTION: This procedure is invoked when a DM
* frame is received. When a DM frame is found, no
* actions are undertaken. The DM response frame is
* used to indicate a temporary inactive DCE or DTE.
* CHANNEL
* PASSED VARIABLES:
* RETURNS: None
* GLOBAL VARIABLES USED: None
* GLOBAL VARIABLES CHANGED: None
* MODULES CALLED: None
* CALLING MODULES: ROUTE$IN
* AUTHOR: Mark Weber
* HISTORY: 1.0 Mark Weber -30 AUG 85- original translated PL/M
* version
*****
RCV$DM: PROCEDURE(CHANNEL, P$BIT)PUBLIC;

    DECLARE (CHANNEL, P$BIT) BYTE;
    DECLARE TP$82A(*) BYTE DATA(CR,LF, CHANNEL A');
    'TP$82A
    ENTERING RCV$DM, CHANNEL A';
    DECLARE TP$82B(*) BYTE DATA(CR,LF, CHANNEL B');
    'TP$82B
    ENTERING RCV$DM, CHANNEL B';

    IF (CHANNEL = 1 OR CHANNEL = 2) THEN
        DO CASE CHANNEL:
            /*CASE 0 IS NULL */
            DO: /*CASE 1 IS FOR CHANNEL A */
                CALL SENDSEQ(@TP$82A, LENGTH(TP$82A));
                CALL SRVC$TAB$H$SKP(1, I$FRAME$SIZE);
            END; /*CASE 1 */
            DO: /*CASE 2 IS FOR CHANNEL B */
                CALL SENDSEQ(@TP$82B, LENGTH(TP$82B));
                CALL SRVC$TAB$H$SKP(2, I$FRAME$SIZE);
            END; /*CASE 2 */
        END; /* END CASE */
    END;
END RCV$DM;

/*****
* DATE: 29 AUG 85
*****

```

SOURCE CODE FOR LAP80.SRC MODULE TO NETX.25, 5 NOV 85

```

* VERSION: 1.0
* NAME: RCV$UA
* MODULE NUMBER: 8.3
* DESCRIPTION: This procedure processes received UA
*              frames. When a UA frame is received the last U frame
*              command received is acted upon. The p$bit is reserved
*              for future use in error detection and correction.
*
* PASSED VARIABLES: CHANNEL, P$BIT
* RETURNS: None
* GLOBAL VARIABLES USED: RNR$MODE$A, RNR$MODE$B
* GLOBAL VARIABLES CHANGED: RNR$MODE$A, RNR$MODE$B
* MODULES CALLED: INITACK, SRVC$TAB$H$SKP, SENDSEQ,
*                  FIND$US$CMD
* CALLING MODULES: RCV$U$FRAME, START$DM$MODE
* AUTHOR: Mark Weber
* HISTORY: 1.0 Mark Weber -29 AUG 85-original PL/M version.
*
RCV$UA: PROCEDURE(CHANNEL, F$BIT) PUBLIC REENTRANT;
*****/

DECLARE (CHANNEL, F$BIT) BYTE;
DECLARE TP$76(*) BYTE DATA(CR,LF,
    'TP$76
    ERROR IN RCV$UA
    CHANNEL NUMBER');
DECLARE TP$76A(*) BYTE DATA(CR,LF,
    'TP$76A
    ENTERING RCV$UA,
    CHANNEL A');
DECLARE TP$76B(*) BYTE DATA(CR,LF,
    'TP$76B
    ENTERING RCV$UA,
    CHANNEL B');

IF ((CHANNEL = 1) OR (CHANNEL = 2)) THEN
DO CASE CHANNEL:
/* CASE 0 IS NULL */
DO:
CALL SENDSEQ(@TP$76A, LENGTH(TP$76A));
CALL SRVC$TAB$H$SKP(1, 1$FRAME$SIZE);
CALL FIND$US$CMD(1);
RNR$MODE$A = FALSE;
END; /* END CASE 1 */
DO: /* CASE 2 */
CALL SENDSEQ(@TP$76B, LENGTH(TP$76B));
CALL SRVC$TAB$H$SKP(2, 1$FRAME$SIZE);
CALL FIND$US$CMD(2);
RNR$MODE$B = FALSE;
END; /* END CASE 2 */
/* END CASE */
ELSE CALL SENDSEQ(@TP$76, LENGTH(TP$76)); /* ERROR IN CHANNEL NUMBER */
END RCV$UA;

/*****
* DATE: 29 AUG 85
* VERSION: 1.0
*****/

```

SOURCE CODE FOR LAPBO.SRC MODULE TO NETX.25, 5 NOV 85

```

* NAME: RCV$REJ
* MODULE NUMBER: 9.2.4
* DESCRIPTION: This procedure processes received REJ
*              frames. When a REJ frame is received, a UA frame is
*              send out. The REJ command is then acted upon. The
*              SEND$STATE variable is updated and, if in the RNR$MODE,
*              the mode is terminated.
* PASSED VARIABLES: CHANNEL, PSBIT
* RETURNS: None
* GLOBAL VARIABLES USED: TX01NE, TX02NE, RNR$MODE$A, RNR$MODE$B,
*                        NT01TX, NT02TX
* GLOBAL VARIABLES CHANGED: All gloabals used
* MODULES CALLED: INTIACK, SND$CMDR, LD$TAB$H$SKP,
*                  RCV$IN$SEQ, SEND$SEQ, UPDATES$SEND$STATE
* CALLING MODULES: ROUTE$IN
* AUTHOR: Mark Weber
* HISTORY: 1.0 Mark Weber -29 AUG 85-original PL/M version.
*****
RCV$REJ: PROCEDURE(CHANNEL, PSBIT, RCV$SEQ$NUM, ADDR) PUBLIC;

```

```

    DECLARE (CHANNEL, PSBIT, RCV$SEQ$NUM, ADDR) BYTE;
    DECLARE SND$SEQ$NUM BYTE;
    DECLARE TP$77(*) BYTE DATA(CR,LF, CHANNEL NUMBER');
    'TP$77 ERROR IN RCV$REJ, CHANNEL NUMBER');
    DECLARE TP$77A(*) BYTE DATA(CR,LF, CHANNEL A');
    'TP$77A ENTERING RCV$REJ, CHANNEL A');
    DECLARE TP$77B(*) BYTE DATA(CR,LF, CHANNEL B');
    'TP$77B ENTERING RCV$REJ, CHANNEL B');

    IF (ADDR = A$ADD) OR (ADDR = B$ADD) THEN /* SAME ACTION FOR COMMAND AND
                                              RESPONSE, IGNORE THE P/F BIT */
        IF ((CHANNEL = 1) OR (CHANNEL = 2)) THEN
            DO CASE CHANNEL;
            /* CASE 0 IS NULL */
            /* CASE 1 */
            DO:
                CALL SENDSEQ(@TP$77A, LENGTH(TP$77A));
                RNR$MODE$A = FALSE;
                SND$SEQ$NUM = (SHR(NT01TX(TX01NS + 1), 1) AND 07);
                IF ((SEND$STATE$A + 1) = RCV$SEQ$NUM) THEN
                    CALL UPDATES$SEND$STATE(1, RCV$SEQ$NUM);
                ELSE /* FRAME NOT IN QUEUE */
                    DO:
                        CALL SND$CMDR(1,1,NT01TX(TX01NS + 1),2);
                        CALL SRVC$TAB$H$SKP(3, 1$FRAME$SIZE);
                    END;
                CALL SRVC$TAB$H$SKP(1, 1$FRAME$SIZE);
            END;/* END CASE 1 */
            DO;/* CASE 2 */
                CALL SENDSEQ(@TP$77B, LENGTH(TP$77B));
            END;

```


SOURCE CODE FOR LAPBD.SRC MODULE TO NETX.25, 5 NOV 85

```

RNR$MODE$B = FALSE;
SND$SEQ$NUM = (SHR(NT02TX(TX02NS + 1), 1) AND 07);
IF ((SEND$STATE$B + 1) = RCV$SEQ$NUM) THEN
    CALL UPDATE$SEND$STATE(2, RCV$SEQ$NUM);
ELSE /* FRAME NOT IN QUEUE */
DO;
    CALL SND$CMDR(2,1,NT02TX(TX02NS + 1),2);
    CALL SRVC$TAB$H$SKP(4, I$FRAME$SIZE);
END;
CALL SRVC$TAB$H$SKP(2, I$FRAME$SIZE);
END; /* END CASE 2 */
END; /* END CASE */
ELSE CALL SEND$SEQ(@TP$77, LENGTH(TP$77)); /* ERROR IN CHANNEL NUMBER */
END RCV$REJ;

/*****
* DATE: 29 AUG 85
* VERSION: 1.0
* NAME: RCV$RNR
* MODULE NUMBER: 9.2.5
* DESCRIPTION: This procedure processes received RNR
* FRAMES. When a RNR FRAME is received, A UA FRAME is
* send out. The RNR command is then acted upon. The
* SEND$STATE variable is updated and, if in the RNR$MODE,
* the mode is initiated.
* PASSED VARIABLES: CHANNEL, P$BIT
* RETURNS: None
* GLOBAL VARIABLES USED: TX02NE, TX02NE, RNR$MODE$A, RNR$MODE$B,
* NT01TX, NT02TX
* GLOBAL VARIABLES CHANGED: All globals used
* MODULES CALLED: RCV$IN$SEQ, INITACK, SRVC$TAB$H$SKP,
* UPDATE$SEND$STATE
* CALLING MODULES: ROUTE$IN
* AUTHOR: Mark Weber
* HISTORY: 1.0 Mark Weber -29 AUG 85-original PL/M version.
* RCV$RNR: PROCEDURE(CHANNEL, P$BIT, SEQ$NUM, ADDR)PUBLIC;
*****/

DECLARE (CHANNEL, P$BIT, SEQ$NUM, ADDR) BYTE;
DECLARE TP$78(*) BYTE DATA(CR,LF);
ERROR IN RCV$RNR CHANNEL NUMBER';
DECLARE TP$78A(*) BYTE DATA(CR,LF, CHANNEL A');
ENTERING RCV$RNR, CHANNEL A';
DECLARE TP$78B(*) BYTE DATA(CR,LF, CHANNEL B');
ENTERING RCV$RNR, CHANNEL B';

IF (P$BIT = 1) THEN
    IF ((CHANNEL = 1) OR (CHANNEL = 2)) THEN

```

SOURCE CODE FOR LAPBO.SRC MODULE TO NETX.25, 5 NOV 85

```

DO CASE CHANNEL;
  /* CASE 0 IS NULL */
  /* CASE 1 */
DO:
  CALL SENDSEQ(@TP$78A, LENGTH(TP$78A));
  IF (ADDR = A$ADD) THEN
DO:
  RNR$MODE$A = TRUE;
  CALL UPDATE$SEND$STATE(1, SEQ$NUM);
END;
  CALL SRV$TAB$H$SKP(1, I$FRAME$SIZE);
END; /* END CASE 1 */
DO: /* CASE 2 */
  CALL SENDSEQ(@TP$78B, LENGTH(TP$78B));
  IF (ADDR = B$ADD) THEN
DO:
  RNR$MODE$B = TRUE;
  CALL UPDATE$SEND$STATE(2, SEQ$NUM);
END;
  CALL SRV$TAB$H$SKP(2, I$FRAME$SIZE);
END; /* END CASE 2 */
ELSE CALL SENDSEQ(@TP$78, LENGTH(TP$78)); /* ERROR IN CHANNEL NUMBER */
END RCV$RNR;
/*****
* DATE: 29 AUG 85
* VERSION: 1.0
* NAME: RCV$RNR
* MODULE NUMBER: 9.2.6
* DESCRIPTION: This procedure processes received RR
*              frames. When a RR frame is received, a UA frame is
*              sent out. The RNR command is then acted upon. The
*              SEND$STATE variable is updated and, if in the RNR$MODE,
*              the mode is initiated.
*              CHANNEL, P$BIT
* PASSED VARIABLES:
* RETURNS: None
* GLOBAL VARIABLES USED: RNR$MODE$A, RCV$STATES, RNR$MODE$B,
*              RCV$STATES$B
* GLOBAL VARIABLES CHANGED: All globals used
* MODULES CALLED: INITACK, SND$CMDR, LD$TAB$H$SKP
* CALLING MODULES: ROUTES$IN
* AUTHOR: Mark Weber
* HISTORY: 1.0 Mark Weber -29 AUG 85-original PL/M version.
* *****/
RCV$RNR: PROCEDURE(CHANNEL, P$BIT, SEQ$NUM, ADDR)PUBLIC;
  DECLARE (CHANNEL, P$BIT, SEQ$NUM, ADDR)BYTE;
  DECLARE TP$79(*) BYTE DATA(CR,LF,
  'TP$79
  ERROR IN RCV$RNR CHANNEL NUMBER');
  DECLARE TP$79A(*) BYTE DATA(CR,LF,

```

SOURCE CODE FOR LAPBO.SRC MODULE TO NETX.25, 5 NOV 85

```

      'TP$79A      ENTERING RCV$RR,      CHANNEL A');
      DECLARE TP$79B(*) BYTE DATA(CR,LF, CHANNEL A');
      'TP$79B      ENTERING RCV$RR,      CHANNEL B');

      IF ((CHANNEL = 1) OR (CHANNEL = 2)) THEN
      DO CASE CHANNEL;
      /* CASE 0 IS NULL */
      /* CASE 1 */
      DO;
      CALL SENDSEQ(@TP$79A, LENGTH(TP$79A));
      IF (ADDR = A$ADD) THEN
      DO;
      RNR$MODE$A = FALSE;
      CALL UPDATE$SEND$STATE(1, SEQ$NUM);
      END;
      CALL SRVC$TAB$H$SKP(1, I$FRAME$SIZE);
      END; /* END CASE 1 */
      /* CASE 2 */
      DO; /* CASE 2 */
      CALL SENDSEQ(@TP$79B, LENGTH(TP$79B));
      IF (ADDR = B$ADD) THEN
      DO;
      RNR$MODE$B = FALSE;
      CALL UPDATE$SEND$STATE(2, SEQ$NUM);
      END;
      CALL SRVC$TAB$H$SKP(2, I$FRAME$SIZE);
      END; /* END CASE 2 */
      ELSE CALL SENDSEQ(@TP$79, LENGTH(TP$79)); /* ERROR IN CHANNEL NUMBER */
      END RCV$RR;

```

```

/*****
* DATE: 29 AUG 85
* VERSION: 1.0
* NAME: RCV$DISC
* MODULE NUMBER: 9.2.7
* DESCRIPTION: This procedure processes received DISC
*              FRAMES. When a DISC FRAME is receive, program control
*              is returned to the main program where the DM mode will
*              be entered.
*
* PASSED VARIABLES: CHANNEL, P$BIT
* RETURNS: None
* GLOBAL VARIABLES USED: TX01NE, TX02NE, RNR$MODE$A, RNR$MODE$B,
*              SABM$MODE$A, SABM$MODE$B, SEND$STATE$A,
*              SEND$STATE$B, RCV$STATE$A, RCV$STATE$B,
*              NT01NE, NT02NE, TX01NS, TX02NS, NT01NS, NT02NS
* GLOBAL VARIABLES CHANGED: TX01NE, TX02NE
* MODULES CALLED: SENDSEQ
* CALLING MODULES: ROUTE$IN
* AUTHOR: Mark Weber
*****/

```

SOURCE CODE FOR LAPBO.SRC MODULE TO NETX.25, 5 NOV 85

```

* HISTORY: 1.0 Mark Weber -29 AUG 85-original PL/M version.
*****
RCV$DISC: PROCEDURE(CHANNEL, P$BIT)PUBLIC;
*****

DECLARE (CHANNEL, P$BIT) BYTE;
DECLARE TP$80(*) BYTE DATA(CR,LF,
    'TP$80' ERROR IN RCV$DISC, CHANNEL NUMBER');
DECLARE TP$80A(*) BYTE DATA(CR,LF,
    'TP$80A' ENTERING RCV$DISC, CHANNEL A');
DECLARE TP$80B(*) BYTE DATA(CR,LF,
    'TP$80B' ENTERING RCV$DISC, CHANNEL B');

IF ((CHANNEL = 1) OR (CHANNEL = 2)) THEN
DO CASE CHANNEL;
: /* CASE 0 IS NULL */
DO; /* CASE 1 */
CALL SENDSEQ(@TP$80A, LENGTH(TP$80A));
IF (P$BIT = 1) THEN CALL SND$UA(1,1);
RNR$MODE$A = FALSE;
SABM$MODE$A = FALSE;
RCV$STATE$A = 0;
SEND$STATE$A = 0;
NT01NE = 0;
NT01NS = 0;
TX01NE = 0;
TX01NS = 0;
END; /* END CASE 1 */
DO; /* CASE 2 */
CALL SENDSEQ(@TP$80B, LENGTH(TP$80B));
IF (P$BIT = 1) THEN CALL SND$UA(2,1);
RNR$MODE$B = FALSE;
SABM$MODE$B = FALSE;
RCV$STATE$B = 0;
SEND$STATE$B = 0;
NT02NE = 0;
NT02NS = 0;
TX02NE = 0;
TX02NS = 0;
END; /* END CASE 2 */
ELSE CALL SENDSEQ(@TP$80, LENGTH(TP$80)); /* ERROR IN CHANNEL NUMBER */
END RCV$DISC;

/*****
* DATE: 29 AUG 85
* VERSION: 1.0
* NAME: RCV$CMDR
* MODULE NUMBER: 9.2.9
* DESCRIPTION: This procedure processes received CMDR
*****/

```

SOURCE CODE FOR LAPB0.SRC MODULE TO NETX.25, 5 NOV 85

```

*
* FRAMES. When a CMDR FRAME is receive, program deter-
* mines the appropriate error recovery actions. This
* actions are not specifically stated in the X.25 rec-
* commendation. Presently only limited error recovery
* occurs.
*
* PASSED VARIABLES: CHANNEL, PSBIT
* RETURNS: None
* GLOBAL VARIABLES USED: SABM$MODE$A, SABM$MODE$B
* GLOBAL VARIABLES CHANGED: SABM$MODE$A, SABM$MODE$B
* MODULES CALLED: SRVC$TAB$H$SKP
* CALLING MODULES: ROUTES$IN
*
* AUTHOR: Mark Weber
* HISTORY: 1.0 Mark Weber -29 AUG 85-original PL/M version.
*
* RCV$CMDR: PROCEDURE(CHANNEL, PSBIT)PUBLIC;
*****/

DECLARE (CHANNEL, PSBIT) BYTE;
DECLARE (ERR$CHA, ERR$CHB) BYTE;

IF ((CHANNEL = 1) OR (CHANNEL = 2)) THEN
DO CASE CHANNEL;
/* CASE 0 IS NULL */
DO;
/* CASE 1 */
CALL SENDSEQ(@TP$75A, LENGTH(TP$75A));
ERR$CHA = (NT01RX(NT01NS + 4) AND OFH);
IF (ERR$CHA AND 01H) THEN SABM$MODE$A = FALSE;
IF (ERR$CHA AND 02H) THEN SABM$MODE$A = FALSE;
IF (ERR$CHA AND 04H) THEN SABM$MODE$A = FALSE;
IF (ERR$CHA AND 08H) THEN SABM$MODE$A = FALSE;
CALL SRVC$TAB$H$SKP(1, 1$FRAME$SIZE);
END; /* END CASE 1 */
/* CASE 2 */
DO;
CALL SENDSEQ(@TP$75B, LENGTH(TP$75B));
ERR$CHB = (NT02RX(NT02NS + 4) AND OFH);
IF (ERR$CHB AND 01H) THEN SABM$MODE$B = FALSE;
IF (ERR$CHB AND 02H) THEN SABM$MODE$B = FALSE;
IF (ERR$CHB AND 04H) THEN SABM$MODE$B = FALSE;
IF (ERR$CHB AND 08H) THEN SABM$MODE$B = FALSE;
CALL SRVC$TAB$H$SKP(2, 1$FRAME$SIZE);
END; /* END CASE 2 */
END; /* END CASE */
ELSE CALL SENDSEQ(@TP$75, LENGTH(TP$75)); /* ERROR IN CHANNEL NUMBER */
END RCV$CMDR;

END LAPB$MODULE;

/***** THE END *****/

```

3. LAPB1.SRC - Module with LAP B procedures

```

SOURCE CODE FOR LAPB1.SRC MODULE TO NETX25.SRC, 19 NOV 85

$TITLE('UNID II NETWORK TEST PROGRAM, MODULE LAPB1, 19 NOV 85')
$XREF OPTIMIZE(2)
/*****
* DATE: 19 NOV 85
* VERSION: 1.1
* TITLE: ISO layer 2 LAP B simulation
* FILENAME: LAPB1.SRC
* COORDINATOR: Capt Mark W. Weber
* PROJECT: UNIT II
* OPERATING SYSTEM: INTEL SYSTEM III/230
* LANGUAGE: PL/M 86
* USE: This file requires no includes, but must be linked with LAPB0.OBJ
*      NETX25.OBJ, PKCT.OBJ, and SMALL.LIB. The program may be linked
*      and located with the SUBMIT file NETX25.CSD.
* CONTENTS:
*      STCTCA - simulates a timer for channel A
*      STCTCB - simulates a timer for channel B
*      TIMESDELAY$CHA - simulates "time-out" on channel A
*      TIMESDELAY$CHB - simulates "time-out" on channel B
*      DET$DESTONE - determines destinations for channel A
*      DET$DESTTWO - determines destinations for channel B
*      DET$DESTLN - determines the destinations of local messages
*      READ$TAB - reads the packets moved to the NTLCTB table
*      FIND$IFRAME - determines if more than one I FRAME is in the
*                  the transmit table
* FUNCTION: Simulates the operation of the SBC 88/45 in the UNID II using
*           the datalink layer as described by CCITT recommendation X.25.
*           ISIS calls are used to provide a user interface and to trace
*           the process flow of the LAP B protocol employed by the CCITT
*           X.25 recommendation.
* HISTORY: 1.1 Mark Weber - implemented X.25 datalink procedures
*          1.0 TRANSLATED 5 APR 84 by C.T. Chrildress
*****/

```

```

LAPB1$MODULE: DO:

```

```

/***** EXTERNAL PROCEDURES FOR ISIS SYSTEM CALLS *****/

```

```

FRAMES$PRESENT: PROCEDURE (CHANNEL) BYTE EXTERNAL;
                 DECLARE CHANNEL BYTE;
                 END FRAMES$PRESENT;

```

```

PRINTI: PROCEDURE (NUM) EXTERNAL;
          DECLARE NUM INTEGER;
          END PRINTI;

```

```

DSPLY$FRAME$HDR: PROCEDURE (CHANNEL) EXTERNAL;
                  DECLARE CHANNEL BYTE;

```

SOURCE CODE FOR LAPB1.SRC MODULE TO NETX25.SRC, 19 NOV 85

```

END D$PLY$FRAME$HDR;

SENDSEQ:
  PROCEDURE (MSG, TOTAL) EXTERNAL;
  DECLARE MSG POINTER, TOTAL WORD;
  END SENDSEQ;

SRVC$TAB$H$SKP:
  PROCEDURE (TABLE, FRAME$SIZE) EXTERNAL;
  DECLARE TABLE BYTE, FRAME$SIZE INTEGER;
  END SRVC$TAB$H$SKP;

DECLARE CR          LITERALLY 'ODH',
LF                LITERALLY 'OAH';
DECLARE CR$LF (2) BYTE DATA (ODH, OAH);

/***** END EXTERNALS *****/

DECLARE
  FALSE          LITERALLY 'OH', /* USE AS FLAGS TO TEST */
  TRUE           LITERALLY 'OFFH', /* BITS FOR BRANCHING */
  CONCTC         LITERALLY 'ODSH', /* NETWORK MONITOR CTC PORT ADDRESS */
  CONCND         LITERALLY 'ODFH', /* NETWORK MONITOR USRT COMMAND PORT ADDRESS */
  CONDAT         LITERALLY 'ODEH', /* NETWORK MONITOR USRT DATA PORT ADDRESS */

  NET$RI$DE$T$ERR LITERALLY '10', /* NET ROUTE$IN DEST ERROR ENTRY */
  NET$RO$DE$T$ERR LITERALLY '11', /* NET ROUTE$OUT DEST ERROR ENTRY */
  PACKET$SIZE    LITERALLY '138', /* PACKET IS 138 BYTE BLOCK */
  PACKET$IN$TABLE LITERALLY '10', /* # PACKETS IN TABLE */
  PACKET$TABLE$SIZE LITERALLY '1380',
  I$FRAME$SIZE   LITERALLY '140',
  U$FRAME$SIZE   LITERALLY '2',
  S$FRAME$SIZE   LITERALLY '2',
  CMOR$FRAME$SIZE LITERALLY '5',
  FRAME$IN$TABLE LITERALLY '10',
  FRAME$TABLE$SIZE LITERALLY '1400',
  DATA$SIZE     LITERALLY '128',
  IP$DATA$SIZE   LITERALLY '96',
  TCP$DATA$SIZE  LITERALLY '72';

DECLARE READY    LITERALLY '1',
DONE            LITERALLY '2';

DECLARE SABM$MODE$A BYTE EXTERNAL,
SABM$MODE$B      BYTE EXTERNAL;

/* NETWORK VARIABLES FOR THIS UNIT */

/* NOTES: 1. THIS$UNITID$NBR MUST REFLECT WHICH UNIT THIS IS.
2. THIS$COUNTRY$CODE MUST REFLECT THE AREA TO WHICH THIS
```


SOURCE CODE FOR LAPB1.SRC MODULE TO NETX25.SRC, 19 NOV 85

```

UNID IS LOCATED.
3. MAX$COUNTRY$CODE WILL INDICATE WHICH COUNTRY CODES
   ARE CURRENTLY OPERATIONAL. CC = 0000 IS RESERVED
   FOR THE DELNET MONITOR.
4. MAX$NETWORK$CODE WILL INDICATE HOW MANY UNIDS ARE
   CURRENTLY OPERATIONAL WITHIN A PARTICULAR COUNTRY.
5. FOR DETAILED INFORMATION ON THE ABOVE, REFER TO
   PHISTER'S THESIS, APPENDIX D.
*/

```

```

DECLARE
  THIS$UNID$NBR LITERALLY '02H', /* UNIQUE ADDRESS OF THIS UNID */
  THIS$COUNTRY$CODE LITERALLY '01H', /* COUNTRY WHERE THIS UNID RESIDES */
  MAX$COUNTRY$CODE LITERALLY '01H', /* COUNTRIES CURRENTLY OPERATIONAL */
  MAX$NETWORK$CODE LITERALLY '03H', /* NUMBER OF UNIDS OPERATIONAL IN CC */
*/

```

```

STAT$NBR LITERALLY '20', /* NUMBER OF ENTRIES IN STATUS TABLE */

/* VARIABLES USED IN N.MAIN$U2 AND N.INSIO$U2 */
CTCNOA BYTE EXTERNAL,
/* PROGRESSIVE NUMBER OF TIME COUNTS FOR NETWORK CHANNEL A */
CTCNOB BYTE EXTERNAL,
/* PROGRESSIVE NUMBER OF TIME COUNTS FOR NETWORK CHANNEL B */
MAXNOA LITERALLY '2', /* PREVIOUSLY 64H = 100D */
/* MAXIMUM NUMBER OF TIMING COUNTS FOR NETWORK CHANNEL A */
MAXNOB LITERALLY '2', /* PREVIOUSLY 64H = 100D */
/* MAXIMUM NUMBER OF TIMING COUNTS FOR NETWORK CHANNEL B */
RETRAN$A BYTE,
/* PROGRESSIVE NUMBER OF RETRANSMISSIONS OF A FRAME */
RETRAN$B BYTE,
/* PROGRESSIVE NUMBER OF RETRANSMISSIONS OF A FRAME */
MAXRETRAN$A LITERALLY '6',
/* MAXIMUM NUMBER OF RETRANSMISSIONS OF A FRAME */
MAXRETRAN$B LITERALLY '6',
/* MAXIMUM NUMBER OF RETRANSMISSIONS OF A FRAME */

```

```

DECLARE BUSYSTATUS LITERALLY 'OFFH',
NMBR$MSK LITERALLY '07H',
ESC LITERALLY '1BH',
EOT LITERALLY '04H';
DECLARE ASCII(16) BYTE EXTERNAL;
DECLARE MSGNUM BYTE EXTERNAL;
DECLARE OUT$TAB$FULL BYTE EXTERNAL;
DECLARE I$CNTL LITERALLY '01H';

```

```

/***** GLOBAL CONSTANTS USED BY LAP B POCESSES *****/
/*****
/* DEFINITIONS FOR THE NETWORK SERIAL INPUT/OUTPUT USARTS */
*/

```

SOURCE CODE FOR LAPB1.SRC MODULE TO NETX25.SRC, 19 NOV 85

```

/*****
DECLARE  BAUD$LSB      BYTE,
        BAUD$MSB      BYTE,
        NUM$BYTESSENT INTEGER;
        /* MORE TO BE ADDED LATER */
/*****
/***** ADDITIONAL GENERAL DECLARES NEEDED FOR THIS PROGRAM *****/
/*****
DECLARE  TIMCHA      BYTE EXTERNAL,
        TIMCHB      BYTE EXTERNAL,
        HSKP$ERR    INTEGER EXTERNAL;
        /* MORE TO BE ADDED LATER */
/*****
/***** DATA TABLES USED IN THIS PROGRAM *****/
/*****
DECLARE  NT01RX(FRAME$TABLE$SIZE)  BYTE EXTERNAL,
        NT01NS INTEGER EXTERNAL,
        NT01NE INTEGER EXTERNAL,
        NT01SZ INTEGER EXTERNAL,
        NT02RX(FRAME$TABLE$SIZE)  BYTE EXTERNAL,
        NT02NS INTEGER EXTERNAL,
        NT02NE INTEGER EXTERNAL,
        NT02SZ INTEGER EXTERNAL,
        NT01TX(FRAME$TABLE$SIZE)  BYTE EXTERNAL,
        TX01NS INTEGER EXTERNAL,
        TX01NE INTEGER EXTERNAL,
        TX01SZ INTEGER EXTERNAL,
        NT02TX(FRAME$TABLE$SIZE)  BYTE EXTERNAL,
        TX02NS INTEGER EXTERNAL,
        TX02NE INTEGER EXTERNAL,
        TX02SZ INTEGER EXTERNAL,
        I$FRAME$QUE(FRAME$TABLE$SIZE)  BYTE EXTERNAL,
        I$FRAME$QUE$NS INTEGER EXTERNAL,
        I$FRAME$QUE$NE INTEGER EXTERNAL,
        I$FRAME$QUE$SZ INTEGER EXTERNAL,
        LCNTTB(PACKET$TABLE$SIZE)  BYTE EXTERNAL,
        LCNTNS INTEGER EXTERNAL,

```

SOURCE CODE FOR LAPB1.SRC MODULE TO NETX25.SRC, 19 NOV 85

```

LCNTNE INTEGER EXTERNAL,
LCNTSZ INTEGER EXTERNAL,

NLCTB(PACKET$TABLESIZE) BYTE EXTERNAL,
NLCONS INTEGER EXTERNAL,
NLCLNE INTEGER EXTERNAL,
NLCSZ INTEGER EXTERNAL,

STATTB(STAT$NBR) BYTE:

DECLARE
  (LSEM$1, LSEM$2, LSEM$3, /* LOCAL TO NET SEMAPHORE AND */
   LSEM$4, NSEM$1, NSEM$2) BYTE EXTERNAL, /* NET TO LOCAL SEMAPHORE */

  (LPTR$1, LSPARE$1, LPTR$2, LSPARE$2, /* LOC TO NET PACKET PTR */
   LPTR$3, LSPARE$3, LPTR$4, LSPARE$4, /* LOC TO NET PACKET PTR */
   NPTR$1, NSPARE$1, NPTR$2, NSPARE$2) POINTER EXTERNAL; /* NET TO LOC PACKET PTR */

```

/* MISCELLANEOUS DECLARATIONS */

```

DECLARE
  FOREVER BYTE EXTERNAL,
  DESTINATION WORD EXTERNAL,
  SEQ$NUM$A BYTE EXTERNAL, /* DESTINATION OF A FRAME */
  SEQ$NUM$B BYTE EXTERNAL, /* ACKNOWLEDGE VARIABLE DECLARATIONS */
  SEND$STATE$A BYTE EXTERNAL,
  SEND$STATE$B BYTE EXTERNAL,
  RCV$STATE$A BYTE EXTERNAL,
  RCV$STATE$B BYTE EXTERNAL,

  TP$30(*) BYTE DATA(CR,LF,
    DESTINATION OF NT01RX IS NL');
  TP$31(*) BYTE DATA(CR,LF,
    DESTINATION OF NT01RX IS NN');
  TP$32(*) BYTE DATA(CR,LF,
    DESTINATION OF NT02RX IS NL');
  TP$33(*) BYTE DATA(CR,LF,
    DESTINATION OF NT02RX IS NN');
  TP$34(*) BYTE DATA(CR,LF,
    DESTINATION$UNID >= THIS$UNID$NBR');
  TP$35(*) BYTE DATA(CR,LF,
    DESTINATION$UNID < THIS$UNID$NBR');
  TP$36(*) BYTE DATA(CR,LF,
    DESTINATION IS CHANNEL A');
  TP$37(*) BYTE DATA(CR,LF,
    DESTINATION IS CHANNEL B');
  TP$55(*) BYTE DATA(CR,LF,
    READING NLCTB PACKET TO CONJUIT');

```

SOURCE CODE FOR LAPB1.SRC MODULE TO NETX25.SRC, 19 NOV 85

```

/*****
/*****
* DATE: 30 SEP 84
* VERSION: 1.0
* NAME: STCTCA
* MODULE NUMBER: 9.4.1
* DESCRIPTION: This is a dummy module for the TI timer
*
* for channel A
* PASSED VARIABLES: None
* RETURNS: None
* GLOBAL VARIABLES USED: TIMCHA
* GLOBAL VARIABLES CHANGED: TIMCHA
* MODULES CALLED: None
* CALLING MODULES: None
* AUTHOR: C.T. Childress
* HISTORY: 1.0 Capt C.T. Childress - original translated PL/M
*
*****/
STCTCA: PROCEDURE PUBLIC;

```

TIMCHA = FALSE;

END STCTCA;

```

/*****
* DATE: 30 SEP 85
* VERSION: 1.0
* NAME: STCTCB
* MODULE NUMBER: 9.4.2
* DESCRIPTION: This is a dummy module for the TI timer
*
* for channel B.
* PASSED VARIABLES: None
* RETURNS: None
* GLOBAL VARIABLES USED: TIMCHB
* GLOBAL VARIABLES CHANGED: TIMCHB
* MODULES CALLED: None
* CALLING MODULES: None
* AUTHOR: T. C. Childress
* HISTORY: 1.0 Capt C.T. Childress - original translated PL/M
*
*****/
STCTCB: PROCEDURE PUBLIC;

```

TIMCHB = FALSE;

END STCTCB;

```

/*****
NOTES: 1. SEE PROCEDURE STCTCA FOR AN EXAMPLE OF CALCULATING THE APPROPRIATE
TIME DELAY.

```

SOURCE CODE FOR LAPB1.SRC MODULE TO NETX25.SRC, 19 NOV 85

```

2. VARIABLES TOUTCA, CTCNOA, AND MAXNOA ALLOW CHANNEL A TO BE INDEPENDENT
OF CHANNEL B WHEN CONSIDERING TIME DELAY TO AN ADJACENT UNIT.
/*****
*
* DATE: 30 SEP 84
* VERSION: 1.0
* NAME: TIMES$DELAY$CHA
* MODULE NUMBER: 9.4
* DESCRIPTION: The purpose of this procedure is to
* produce a time delay to be used between successive
* transmissions out of CH A of I frames when a valid ACK
* has not been received. The procedure begins by calling
* the (assemble routine STCTCA which it tailizes and
* starts the CTC channel #x. Each time the CTC runs out
* its interrupt routine (TOUTCA) increments the global
* variable 'CTCNOA'. When ever the CTCNOA value is
* greater or equal to the global variable 'MAXNOA', the
* global variable 'TIMCHA' is changed to TRUE.
*
* PASSED VARIABLES: None
* RETURNS: None
* GLOBAL VARIABLES USED: STCTCA, TIMCHA, MAXNOA
* GLOBAL VARIABLES CHANGED: STCTCA, TIMCHA
* MODULES CALLED: STCTCA - in module N.INSIO$U2
* CALLING MODULES: ROUTE$OUT
* AUTHOR: C.T. Childress
* HISTORY: 1.0 Capt C.T. Childress - original translated PL/M
* version
*****/
TIMES$DELAY$CHA: PROCEDURE PUBLIC REENTRANT;
*****
IF TIMCHA = TRUE THEN
  CALL STCTCA; /* IF TIMER IS NOT RUNNING, THEN START IT */
ELSE
  CTCNOA = CTCNOA + 1;
  IF CTCNOA >= MAXNOA THEN DO;
    TIMCHA = TRUE;
    CTCNOA = 0;
  END;
END TIMES$DELAY$CHA;
/*****
*
* DATE: 30 SEP 84
* VERSION: 1.0
* NAME: TIMES$DELAY$CHB
* MODULE NUMBER: 9.6
* DESCRIPTION: The purpose of this procedure is to
* produce a time delay to be used between successive
* transmissions out of CH B of I frames when a valid ACK
* has not been received. The procedure begins by calling
* the (assemble routine STCTCB which it tailizes and

```

SOURCE CODE FOR LAPB1.SRC MODULE TO NETX25.SRC. 19 NOV 85

```

* starts the CTC channel #x. Each time the CTC runs out
* its interrupt routine (TOUTCB) increments the global
* variable 'CTCNOA'. When ever the CTCNOA value is
* greater or equal to the global variable 'MAXNOA', the
* global variable 'TIMCHB' is changed to TRUE.
*
* PASSED VARIABLES:
*   None
* RETURNS:
*   None
* GLOBAL VARIABLES USED:
*   STCTCB, TIMCHB, MAXNOB
* GLOBAL VARIABLES CHANGED:
*   STCTCB, TIMCHB
* MODULES CALLED:
*   STCTCB - in module N.INSIO$U2
* CALLING MODULES:
*   ROUTESOUT
*
* AUTHOR: C.T. Childress
* HISTORY: 1.0 Capt C.T. Childress - original translated PL/M
*
*****
TIMESDELAY$CHB: PROCEDURE PUBLIC REENTRANT;
*****

```

```

IF TIMCHB = TRUE THEN
  CALL STCTCB; /* IF TIMER IS NOT RUNNING, THEN START IT */
ELSE

```

```

  CTCNOB = CTCNOB + 1;
  IF CTCNOB >= MAXNOB THEN DO;
    TIMCHB = TRUE;
    CTCNOB = 0;
  END;

```

```

END TIMESDELAY$CHB;

```

```

/*****
* DATE: 28 AUG 85
* VERSION: 1.1
* NAME: DET$DEST$ONE
* MODULE NUMBER: 8.7
* DESCRIPTION: This procedure looks at the first byte of
* of the network inout table (NT01RX) srvcing CH A and
* determines if the data frame is destined for this
* particular UNID or some other UNID. If it goes to
* this UNID, then 'NL' is returned, if to some other
* UNID then 'NN' is returned. Currently all S frames
* return 'NL' as their destination
*
* PASSED VARIABLES:
*   None
* RETURNS:
*   DESTINATION
* GLOBAL VARIABLES USED:
*   NT01RX, NT0INS
* GLOBAL VARIABLES CHANGED:
*   None
* MODULES CALLED:
*   None
* CALLING MODULES:
*   ROUTESIN
*
* AUTHOR: Mark Weber
* HISTORY: 1.1 Capt Mark Weber - adapted module to new network
* header.
*
* 1.0 Capt C.T. Childress - original translated PL/M
*
*****/

```

SOURCE CODE FOR LAPB1.SRC MODULE TO NETX25.SRC, 19 NOV 85

```

*
* ***** version *****
DET$DEST$ONE: PROCEDURE WORD PUBLIC REENTRANT;
/*
  UNID source and destination network codes are in byte 3 */
  IF (NT01RX (NT01NS + 3) AND OFH) = THIS$UNID$NBR THEN DO;
    CALL SENDSEQ(@TP$30, LENGTH(TP$30));
    DESTINATION = 'NL';
  END;
ELSE DO;
  CALL SENDSEQ(@TP$31, LENGTH(TP$31));
  DESTINATION = 'NN';
END;

RETURN DESTINATION;
END DET$DEST$ONE;

/*****
* DATE: 28 AUG 85
* VERSION: 1.1
* NAME: DET$DEST$TWO
* MODULE NUMBER: 8.8
* DESCRIPTION: This procedure looks at the first byte of
  of the network inout table (NT02RX) srvcing CH B and
  determines if the data frame is destined for this
  particular UNID or some other UNID. If it goes to
  this UNID, then 'NL' is returned, if to some other
  UNID then 'NN' is returned. Currently all S frames
  return 'NL' as their destination
* PASSED VARIABLES: None
* RETURNS: DESTINATION
  GLOBAL VARIABLES USED: NT02RX, NT02NS
* GLOBAL VARIABLES CHANGED: None
* MODULES CALLED: None
* CALLING MODULES: ROUTE$IN
* AUTHOR: Mark Weber
* HISTORY: 1.1 Capt Mark Weber - adapted module to new network
  header.
  1.0 Capt C.T. Childress - original translated PL/M
  version
* *****
DET$DEST$TWO: PROCEDURE WORD PUBLIC REENTRANT;
/*
  UNID source and destination network codes are in byte 3 */
  IF (NT02RX (NT02NS + 3) AND OFH) = THIS$UNID$NBR THEN DO;
    CALL SENDSEQ(@TP$32, LENGTH(TP$32));
    DESTINATION = 'NL';
  END;
ELSE DO;
  CALL SENDSEQ(@TP$33, LENGTH(TP$33));
  DESTINATION = 'NN';
END;

```

SOURCE CODE FOR LAPB1.SRC MODULE TO NETX25.SRC, 19 NOV 85

RETURN DESTINATION;
END DET\$DEST\$TWO;

```

/*****
* DATE: 28 AUG 85
* VERSION: 1.1
* NAME: DET$DEST$LN
* MODULE NUMBER: 8.9
* DESCRIPTION: This module uses a simple algorithm to
* determine the shortest distance between two UNIDs in
* DELNET. Flow control is not considered in the evalu-
* action. The procedure returns a 'CA' if the distance is
* shorter going out the right ring (channel A) or 'CB'
* if the distance is shorter going out the left ring
* (channel B).
* PASSED VARIABLES: None
* RETURNS: DESTINATION
* GLOBAL VARIABLES USED: LCNTNS, LCNTTB,
* GLOBAL VARIABLES CHANGED: None
* MODULES CALLED: SENDSEQ
* CALLING MODULES: ROUTE$IN
* AUTHOR: Mark Weber
* HISTORY: 1.1 Capt Mark Weber - adapted module to new network
* header.
* 1.0 C.T. Childress - 30 SEP 85 - original translated PL/M
* version
*****/
DET$DEST$LN: PROCEDURE WORD PUBLIC REENTRANT;
*****/

DECLARE DESTINATION$UNID INTEGER,
DISTANCE$LEFT INTEGER,
DISTANCE$RIGHT INTEGER;

DESTINATION$UNID = INT(LCNTTB (LCNTNS + 1) AND 0FH);
IF DESTINATION$UNID >= INT(THIS$UNID$NBR) THEN DO;
CALL SENDSEQ(@TP$34, LENGTH(TP$34));
DISTANCE$LEFT = DESTINATION$UNID - INT(THIS$UNID$NBR);
DISTANCE$RIGHT = INT(THIS$UNID$NBR) + (INT(MAX$NETWORK$CODE) -
DESTINATION$UNID + 1);
END;
ELSE DO;
CALL SENDSEQ(@TP$35, LENGTH(TP$35));
DISTANCE$LEFT = INT(THIS$UNID$NBR) +
(INT(MAX$NETWORK$CODE) - DESTINATION$UNID + 1);
DISTANCE$RIGHT = DESTINATION$UNID - INT(THIS$UNID$NBR);
END;
IF DISTANCE$LEFT >= DISTANCE$RIGHT THEN DO;
CALL SENDSEQ(@TP$36, LENGTH(TP$36));
DESTINATION = 'CA';

```


SOURCE CODE FOR LAPB1.SRC MODULE TO NETX25.SRC, 19 NOV 85

```

END;
ELSE DO;
  CALL SENDSEQ(@TP$37, LENGTH(TP$37));
  DESTINATION = 'CB';
END;
RETURN DESTINATION;

END DET$DEST$LN;

```

```

/*****
* DATE: 3 SEP 85
* VERSION: 1.0
* NAME: READ$TAB
* MODULE NUMBER: 9.4
* DESCRIPTION: This procedure reads the test messages
* in the network to local channel table and sends them
* to the system console. This procedure may be used
* in operational software to monitor the message traffic
* on the SBC 88/45.
* PASSED VARIABLES: None
* RETURNS: NTLNE, NTLNS, PACKET$SIZE,
* GLOBAL VARIABLES USED: TCP$DATA$SIZE
* GLOBAL VARIABLES CHANGED: NTLNE
* MODULES CALLED: SRVC$TAB$H$SKP, SEND$SEQ
* CALLING MODULES: START$INFO$XFER
* AUTHOR:
* HISTORY: 1.0 Capt C.T. Childress - original translated PL/M
* version
*****/
READ$TAB: PROCEDURE PUBLIC;

/* IF NSEM$1 = READY THEN
DO;
  MOVB(NPTR$1, @NTLCTB(NTLNE), PACKET$SIZE);
  NSEM$1 = DONE;
  CALL LD$TAB$H$SKP(5, PACKET$SIZE);
END; */

IF ((NTLNE - NTLNS) >= PACKET$SIZE) OR (NTLNS > NTLNE) THEN DO;
  CALL SENDSEQ(@TP$55, LENGTH(TP$55));
  CALL SENDSEQ(@NTLCTB(NTLNS + 66), TCP$DATA$SIZE);
  CALL SRVC$TAB$H$SKP(5, PACKET$SIZE);
END;

END READ$TAB;

/*****
* DATE: 3 SEP 85
*/

```

SOURCE CODE FOR LAPB1.SRC MODULE TO NETX25.SRC, 19 NOV 85

```

* VERSION:
* NAME:
* MODULE NUMBER:
* DESCRIPTION:
*
* This procedure finds the first I
* frame in the tables given. Then the next to service
* pointer for the specified table is placed at the
* location found. The procedure returns a flag indi-
* cating if an I frame was found.
*
* PASSED VARIABLES:
* RETURNS:
* GLOBAL VARIABLES USED:
* GLOBAL VARIABLES CHANGED:
*
* HAVING:
*
* MODULES CALLED:
* CALLING MODULES:
* AUTHOR: C.T. Childress 30 SEP 84
* HISTORY: 1.0 Capt C.T. Childress - original translated PL/M
*
* version
*
* *****
* FIND$I$FRAME: PROCEDURE(TABLE) BYTE PUBLIC REENTRANT;
*
* DECLARE (TABLE,
* FRAME$THERE,
* HAVE$I$FRAME) BYTE;
* DECLARE INDEX INTEGER;
*
* FRAME$THERE = TRUE;
* HAVE$I$FRAME = FALSE;
* INDEX = 0;
*
* DO CASE TABLE;
*
* ; /* ZERO IS NULL */
*
* DO;
* DO WHILE FRAME$THERE;
*
* IF ((TX01NE - TX01NS - INDEX) >= I$FRAME$SIZE) OR
* ((TX01NS + INDEX) > TX01NE) THEN DO;
* IF (NT01TX(TX01NS + INDEX + 1) AND I$CNTL) = 0H
* THEN DO;
* !* HAVE AN INFORMATION FRAME *! *****/
* HAVE$I$FRAME = TRUE;
* FRAME$THERE = FALSE;
* END;
*
* /***** ELSE
* HAVE$I$FRAME = FALSE; !* HAVE AN S FRAME *!
* END;
* ELSE DO;
* HAVE$I$FRAME = FALSE;
*
* *****/

```

SOURCE CODE FOR LAPB1.SRC MODULE TO NETX25.SRC, 19 NOV 85

```

      FRAME$THERE = FALSE;
END;

/****
  INDEX = INDEX + I$FRAME$SIZE;
  END; /* DO WHILE FRAME$THERE */
  END; /* CASE TABLE = 1 */
  *****/

DO;
DO WHILE FRAME$THERE;
  IF ((TX02NE - TX02NS - INDEX) >= I$FRAME$SIZE) OR
    ((TX02NS + INDEX) > TX02NE) THEN DO;
/***** IF (NT02TX(TX02NS + INDEX + 1) AND I$CNTL) = 0H
  THEN DO; /* HAVE AN INFORMATION FRAME */ *****/
    HAVE$IFRAME = TRUE;
    FRAME$THERE = FALSE;
  END;
  ELSE
    HAVE$IFRAME = FALSE; /* HAVE AN S FRAME */
  END;
  END;
  ELSE DO;
    HAVE$IFRAME = FALSE;
    FRAME$THERE = FALSE;
  END;

/****
  INDEX = INDEX + I$FRAME$SIZE;
  END; /* DO WHILE FRAME$THERE */
  END; /* CASE TABLE = 2 */
  *****/

END; /* CASE TABLE */
RETURN HAVE$IFRAME;
END FIND$IFRAME;

END LAPB1$MODULE;
/*****END MODULE LAPB1.SRC*****/

```

4. PCKT.SRC - Module with Packet Layer procedures

```

SOURCE CODE FOR PKT.SRC MODULE TO NETX25.SRC, 19 NOV 85

$TITLE('UNID II NETWORK TEST PROGRAM, MODULE PKT, 19 NOV 85')
$XREF OPTIMIZE(2)
/*****
* DATE: 19 NOV 85
* VERSION: 1.1
* TITLE: ISO layer 2 LAP B simulation
* FILENAME: PKT.SRC
* COORDINATOR: Capt Mark W. Weber
* PROJECT: UNIT II
* OPERATING SYSTEM: INTEL SYSTEM III/230 UNDER ISIS OS
* LANGUAGE: PL/M 86
* USE: This file requires no includes, but must be linked with LAP80.OBJ
* LAPB1.OBJ, NETX25.OBJ, and SMALL.LIB. The program may be linked
* and located with the SUBMIT file NETX25.CSD.
* CONTENTS:
* ROUTE$PACKET - checks the destination byte of the packet
* header
* PACKET$HEADER - non-operative procedure which elaborates on
* the packet header format used by the data link
* simulation. This procedure defines the entry
* point for further packet layer development
* FUNCTION: Simulates the operation of the SBC 88/45 in the UNID II using
* the datalink layer as described by CCITT recommendation X.25.
* ISIS calls are used to provide a user interface and to trace
* the process flow of the LAP B protocol employed by the CCITT
* X.25 recommendation. This module provides the packet layer
* interface for the X.25 recommendation. The manipulation of
* packet header bytes is accomplished through modules within
* this module.
* HISTORY: 1.1 Mark Weber - implemented X.25 semahore usage
* 1.0 TRANSLATED 5 APR 84 by C.T. Chrifidress
*****/

PKT$MODULE: DO;

/***** EXTERNAL PROCEDURES FOR ISIS SYSTEM CALLS *****/

FRAME$PRESENT: PROCEDURE (CHANNEL) BYTE EXTERNAL;
DECLARE CHANNEL BYTE;
END FRAME$PRESENT;

PRINT1: PROCEDURE (NUM) EXTERNAL;
DECLARE NUM INTEGER;
END PRINT1;

DSPLY$FRAME$HDR: PROCEDURE (CHANNEL) EXTERNAL;
DECLARE CHANNEL BYTE;
END DSPLY$FRAME$HDR;

```

```

SENDSEQ:
    PROCEDURE (MSG, TOTAL) EXTERNAL;
    DECLARE MSG POINTER, TOTAL WORD;
    END SENDSEQ;

SRVC$TAB$H$SKP:
    PROCEDURE (TABLE, FRAME$SIZE) EXTERNAL;
    DECLARE TABLE BYTE, FRAME$SIZE INTEGER;
    END SRVC$TAB$H$SKP;

LD$TAB$H$SKP:
    PROCEDURE (TABLE, FRAME$SIZE) EXTERNAL;
    DECLARE TABLE BYTE, FRAME$SIZE INTEGER;
    END LD$TAB$H$SKP;

FIND$I$FRAME:
    PROCEDURE (TABLE) BYTE EXTERNAL;
    DECLARE TABLE BYTE;
    END FIND$I$FRAME;

DET$DEST$LN:
    PROCEDURE WORD EXTERNAL;
    END DET$DEST$LN;

BUILD$I$FRAME:
    PROCEDURE (TABLE$PTR) EXTERNAL;
    DECLARE TABLE$PTR WORD;
    END BUILD$I$FRAME;

DECLARE
    CR          LITERALLY 'ODH',
    LF          LITERALLY 'OAH';
DECLARE CR$LF (2) BYTE DATA (ODH, OAH);

/***** END EXTERNALS *****/

DECLARE
    FALSE      LITERALLY 'OH', /* USE AS FLAGS TO TEST */
    TRUE       LITERALLY 'OFFH', /* BITS FOR BRANCHING */
    CONCTC     LITERALLY 'ODSH', /* NETWORK MONITOR CTC PORT ADDRESS */
    CONCMD     LITERALLY 'ODFH', /* NETWORK MONITOR USART COMMAND PORT ADDRESS */
    CONDAT     LITERALLY 'ODEH', /* NETWORK MONITOR USART DATA PORT ADDRESS */
    NET$RIDE$STERR LITERALLY '10', /* NET ROUTE$IN DEST ERROR ENTRY */
    NET$RODE$STERR LITERALLY '11', /* NET ROUTE$OUT DEST ERROR ENTRY */
    PACKET$SIZE LITERALLY '138', /* PACKET IS 138 BYTE BLOCK */
    PACKET$I$NSTABLE LITERALLY '10', /* # PACKETS IN TABLE */
    I$FRAME$SIZE LITERALLY '1380',
    I$FRAME$SIZE LITERALLY '140',
    U$FRAME$SIZE LITERALLY '2',
    S$FRAME$SIZE LITERALLY '2',
    CMDR$FRAME$SIZE LITERALLY '5',
    FRAME$I$NSTABLE LITERALLY '10',
    FRAME$TABLE$SIZE LITERALLY '1400',

```

SOURCE CODE FOR PKT.SRC MODULE TO NETX25.SRC, 19 NOV 85

```

DATA$SIZE      LITERALLY '128',
IP$DATA$SIZE   LITERALLY '96',
TCP$DATA$SIZE  LITERALLY '72',
PACKET$OFFSET  LITERALLY '2',
READY          LITERALLY '1',
DONE           LITERALLY '2';

```

```

DECLARE  A$ADD  LITERALLY '3',
        B$ADD  LITERALLY '1';

```

/* NETWORK VARIABLES FOR THIS UNIT */

/* NOTES: 1. THIS\$UNID\$NBR MUST REFLECT WHICH UNID THIS IS.
 2. THIS\$COUNTRY\$CODE MUST REFLECT THE AREA TO WHICH THIS UNID IS LOCATED.
 3. MAX\$COUNTRY\$CODE WILL INDICATE WHICH COUNTRY CODES ARE CURRENTLY OPERATIONAL. CC = 0000 IS RESERVED FOR THE DELNET MONITOR.
 4. MAX\$NETWORK\$CODE WILL INDICATE HOW MANY UNIDS ARE CURRENTLY OPERATIONAL WITHIN A PARTICULAR COUNTRY.
 5. FOR DETAILED INFORMATION ON THE ABOVE, REFER TO PHISTER'S THESIS, APPENDIX D.

*/

```

DECLARE
THIS$UNID$NBR LITERALLY '02H', /* UNIQUE ADDRESS OF THIS UNID */
THIS$COUNTRY$CODE LITERALLY '01H', /* COUNTRY WHERE THIS UNID RESIDES */
MAX$COUNTRY$CODE LITERALLY '01H', /* COUNTRIES CURRENTLY OPERATIONAL */
MAX$NETWORK$CODE LITERALLY '03H', /* NUMBER OF UNIDS OPERATIONAL IN CC */

```

```

STAT$NBR  LITERALLY '20', /* NUMBER OF ENTRIES IN STATUS TABLE */

/* VARIABLES USED IN N.MAIN$U2 AND N.INSIO$U2 */
CTCNOA    BYTE EXTERNAL;
/* PROGRESSIVE NUMBER OF TIME COUNTS FOR NETWORK CHANNEL A */
CTCNOB    BYTE EXTERNAL;
/* PROGRESSIVE NUMBER OF TIME COUNTS FOR NETWORK CHANNEL B */
MAXNOA    LITERALLY '2', /* PREVIOUSLY 64H = 100D */
/* MAXIMUM NUMBER OF TIMING COUNTS FOR NETWORK CHANNEL A */
MAXNOB    LITERALLY '2', /* PREVIOUSLY 64H = 100D */
/* MAXIMUM NUMBER OF TIMING COUNTS FOR NETWORK CHANNEL B */
RETRAN$A  BYTE;
/* PROGRESSIVE NUMBER OF RETRANSMISSIONS OF A FRAME */
RETRAN$B  BYTE;
/* PROGRESSIVE NUMBER OF RETRANSMISSIONS OF A FRAME */
MAXRETRAN$A LITERALLY '6', /* PREVIOUSLY 64H = 100D */
/* MAXIMUM NUMBER OF RETRANSMISSIONS OF A FRAME */
MAXRETRAN$B LITERALLY '6', /* PREVIOUSLY 64H = 100D */
/* MAXIMUM NUMBER OF RETRANSMISSIONS OF A FRAME */

```

```

DECLARE BUSYSTATUS LITERALLY 'OFFH',
      NMBR$MSK LITERALLY '07H',
      ESC LITERALLY '1BH',
      EOT LITERALLY '04H';

DECLARE ASCII(16) BYTE EXTERNAL;
DECLARE MSGNUM BYTE EXTERNAL;
DECLARE OUT$TAB$FULL BYTE EXTERNAL;
DECLARE I$CNTL LITERALLY '01H';

/*****
/* ADDITIONAL GENERAL DECLARES NEEDED FOR THIS PROGRAM */
*****/

DECLARE TIMCHA BYTE EXTERNAL,
      TIMCHB BYTE EXTERNAL,
      HSKP$ERR INTEGER EXTERNAL;

/* MORE TO BE ADDED LATER */

/*****
/* DATA TABLES USED IN THIS PROGRAM */
*****/

DECLARE
      NT01RX(FRAME$TABLE$SIZE) BYTE EXTERNAL,
      NT01NS INTEGER EXTERNAL,
      NT01NE INTEGER EXTERNAL,
      NT01SZ INTEGER EXTERNAL,

      NT02RX(FRAME$TABLE$SIZE) BYTE EXTERNAL,
      NT02NS INTEGER EXTERNAL,
      NT02NE INTEGER EXTERNAL,
      NT02SZ INTEGER EXTERNAL,

      NT01TX(FRAME$TABLE$SIZE) BYTE EXTERNAL,
      TX01NS INTEGER EXTERNAL,
      TX01NE INTEGER EXTERNAL,
      TX01SZ INTEGER EXTERNAL,

      NT02TX(FRAME$TABLE$SIZE) BYTE EXTERNAL,
      TX02NS INTEGER EXTERNAL,
      TX02NE INTEGER EXTERNAL,
      TX02SZ INTEGER EXTERNAL,

      LCNTTB(PACKET$TABLE$SIZE) BYTE EXTERNAL,
      LCNTNS INTEGER EXTERNAL,
      LCNTNE INTEGER EXTERNAL,
      LCNTSZ INTEGER EXTERNAL,

      NTLCTB(PACKET$TABLE$SIZE) BYTE EXTERNAL,

```


SOURCE CODE FOR PKT.SRC MODULE TO NETX25.SRC. 19 NOV 85

```

      NTLCSN  INTEGER EXTERNAL,
      NTLCSN  INTEGER EXTERNAL,
      NTLCSZ  INTEGER EXTERNAL,
      STATTB(STAT$NBR)  BYTE;

DECLARE
  (LSEM$1, LSEM$2, LSEM$3,
   LSEM$4, NSEM$1, NSEM$2) BYTE EXTERNAL,
  (LPTR$1, LSPARE$1, LPTR$2, LSPARE$2,
   LPTR$3, LSPARE$3, LPTR$4, LSPARE$4,
   NPTR$1, NSPARE$1, NPTR$2, NSPARE$2) POINTER EXTERNAL;
  /* LOCAL TO NET SEMAPHORE AND */
  /* NET TO LOCAL SEMAPHORE */
  /* LOC TO NET PACKET PTR */
  /* LOC TO NET PACKET PTR */
  /* NET TO LOC PACKET PTR */

```

```

/* MISCELLANEOUS DECLARATIONS */

```

```

DECLARE
  FOREVER BYTE EXTERNAL,
  DESTINATION WORD EXTERNAL,
  SEQNUM$A  BYTE EXTERNAL,
  SEQNUM$B  BYTE EXTERNAL,
  SEND$STATE$A  BYTE EXTERNAL,
  SEND$STATE$B  BYTE EXTERNAL,
  RCV$STATE$A  BYTE EXTERNAL,
  RCV$STATE$B  BYTE EXTERNAL,
  SABMMODE$A  BYTE EXTERNAL,
  SABMMODE$B  BYTE EXTERNAL;
  /* DESTINATION OF A FRAME */
  /* ACKNOWLEDGE VARIABLE DECLARATIONS */

```

```

/* MESSAGES SENT TO TERMINAL */

```

```

DECLARE
  'TP$23  TP$23(*) BYTE PUBLIC DATA(CR,LF,
    PACKET DESTIONED FOR DATA LINK LAYER');
DECLARE
  'TP$24  TP$24(*) BYTE PUBLIC DATA(CR,LF,
    INCOMING PACKET DESTINED FOR CHANNEL A');
DECLARE
  'TP$25  TP$25(*) BYTE PUBLIC DATA(CR,LF,
    INCOMING PACKET DESTINED FOR CHANNEL B');
DECLARE
  'TP$38  TP$38(*) BYTE DATA(CR,LF,
    BUILDING I FRAME FOR CHANNEL A');
DECLARE
  'TP$39  TP$39(*) BYTE DATA(CR,LF,
    BUILDING I FRAME FOR CHANNEL B');

```

```

/*****
* DATE: 28 OCT 85
* VERSION: 1.0
* NAME: ROUTE$PACKET
* MODULE NUMBER: 9.2.11
*****/

```

SOURCE CODE FOR PKT.SRC MODULE TO NETX25.SRC. 19 NOV 85

```

* DESCRIPTION:
* This determines if any of the local
* host ports need servicing. This procedure is called by
* ROUTESIN and represents the packet layer interface to
* to ISO reference model network functions. The present
* algorithm simply checks for a packet in the SBC 544
* receive table simulated by the LCNTTB table and sends
* the packet to one the data link transmit tables.
* Operational software must use the sema-
* phores developed in Appendix E to poll the four SBC 544
* receive tables for messages to send the data link layer.
* The semaphore algorithm is given below in addition to
* algorithm actually used.
*
* PASSED VARIABLES: None
* RETURNS: None
* GLOBAL VARIABLES USED: SABM$MODE$A, SABM$MODE$B, LSEM$1 (2,3,4)
* GLOBAL VARIABLES CHANGED: All globals used
* MODULES CALLED: SENDSEQ, MOVE$HOST$TO$NET
* CALLING MODULES: ROUTE$IN
* ALGORITHM (IMPLEMENTED):
* IF FRAME IN LOCAL TO NETWORK TABLE THEN
* DO; DETERMIN DESTINATION
* IF DESTINATION CHANNEL A THEN
* IF OUT BOUND FRAMES NOT I FRAMES
* DO;
* THEN SEND PACKET TO CHANNEL A TRANSMIT
* BUILD THE I FRAME FOR THE PACKET
* END;
* IF DESTINATION CHANNEL B THEN
* IF OUT BOUND FRAMES NOT I FRAMES
* DO;
* THEN SEND PACKET TO CHANNEL A TRANSMIT
* BUILD THE I FRAME FOR THE PACKET
* END;
*
* ALGORITHM (OPERATIONAL SOFTWARE):
* IF LSEM$1 is READY then
* DO; MOVE$HOST$TO$NET;
* LSEM$1 = DONE;
* END;
* IF LSEM$2 is READY then
* DO; MOVE$HOST$TO$NET;
* LSEM$2 = DONE;
* END;
* IF LSEM$3 is READY then
* DO; MOVE$HOST$TO$NET;
* LSEM$3 = DONE;

```

SOURCE CODE FOR PKT.SRC MODULE TO NETX25.SRC, 19 NOV 85

```

*
*   END;
*   IF LSEM$4 IS READY THEN
*   DO:
*       MOVESHOST$TO$NET;
*       LSEM$4 = DONE;
*   END;
*
*   HISTORY:  1.1 Mark Weber -28 AUG 85-added semaphores
*             1.0 C.T. Childress -30 SEP 84- original translated PL/M
*
* *****
ROUTE$PACKET: PROCEDURE PUBLIC;
* *****
/*-----LOCAL TO NETWORK TABLE PROCESSING-----*/
IF ((LCNTNE - LCNTNS) >= PACKET$SIZE) OR (LCNTNS > LCNTNE) THEN
  IF (SABM$MODE$A AND SABM$MODE$B) THEN
    DO:
      CALL SENDSEQ(@TP$23, LENGTH(TP$23));
      DESTINATION = DET$DEST$LN;
      IF DESTINATION = 'CA' THEN DO: /* PACKET TO CHAN A */
        OUT$TAB$FULL = FIND$IFRAME(1);
        IF (NOT OUT$TAB$FULL) THEN DO:
          CALL SENDSEQ(@TP$24, LENGTH(TP$24));
          CALL BUILD$IFRAME(1);
          CALL SRVC$TAB$H$SKP (6, PACKET$SIZE);
        END; /* IF NOT OUT$ */
      END;
      IF DESTINATION = 'CB' THEN DO: /* PACKET TO CHAN B */
        OUT$TAB$FULL = FIND$IFRAME(2);
        IF (NOT OUT$TAB$FULL) THEN DO:
          CALL SENDSEQ(@TP$25, LENGTH(TP$25));
          CALL BUILD$IFRAME(2);
          CALL SRVC$TAB$H$SKP (6, PACKET$SIZE);
        END; /* IF NOT OUT$ */
      END;
    END;
  END;
  /* END OF (LCNTNE - LCNTNS) CONDITION */
END ROUTE$PACKET;

END PKT$MODULE;
/* *****END MODULE PKT.SRC***** */

```

VITA

Mark William Weber was born on 13 November 1956 in Scottsbluff Nebraska. All primary and secondary education was complete in Gering Nebraska. In August 1975 he entered the University of Idaho as a student in the department of Electrical Engineering. In that same year he entered Air Force ROTC. The following year he received a three year scholarship from the Air Force. December 1979, he graduated with a B.S.E.E. and was commissioned a Second Lieutenant in the United States Air Force. In February 1980, he reviewed his first assignment to Keesler AFB, MS for the nine month Communications - Electronics Course. He completed the program in six months and was then assigned to the 1815 Test Squadron (AFCC) at Scott AFB, IL. While at Scott, he completed the Wideband Systems Evaluation Course at the AFCC Systems Evaluation School and was subsequently assigned as Team Engineer to the Wideband Evaluation Team D. In April 1981, the 1815th Test Squadron was renamed to the 1815th Test and Evaluation Squadron and moved to Wright-Patterson AFB, OH. During his tour with the 1815th Test Squadron, completed overseas evaluations to Greece, Scotland, Japan, Philippines, and Italy. Not long after arriving at Wright-Patterson AFB, Capt Weber became Team Chief for the European Wideband Evaluation Team and remained at that position until reassignment to the Air Force Institute of Technology. His next assignment is OL E, 1842 EEG/CT at Scott AFB IL.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

ADA 164042

REPORT DOCUMENTATION PAGE

1. REPORT SECURITY CLASSIFICATION UNCLASSIFIED			1b. RESTRICTIVE MARKINGS			
2a. SECURITY CLASSIFICATION AUTHORITY			3. DISTRIBUTION/AVAILABILITY OF REPORT			
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE			Approved for public release; distribution unlimited.			
4. PERFORMING ORGANIZATION REPORT NUMBER(S) AFIT/GE/85D-52			5. MONITORING ORGANIZATION REPORT NUMBER(S)			
6a. NAME OF PERFORMING ORGANIZATION School of Engineering		6b. OFFICE SYMBOL (If applicable) AFIT/EN		7a. NAME OF MONITORING ORGANIZATION		
6c. ADDRESS (City, State and ZIP Code) Air Force Institute of Technology Wright Patterson AFB, OH 45433				7b. ADDRESS (City, State and ZIP Code)		
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Rome Air Development Ctr		8b. OFFICE SYMBOL (If applicable)		9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER		
8c. ADDRESS (City, State and ZIP Code) Griffiss AFB NY 13441		10. SOURCE OF FUNDING NOS.				
11. TITLE (Include Security Classification) See Box 19		PROGRAM ELEMENT NO.		PROJECT NO.	TASK NO.	
12. PERSONAL AUTHOR(S) See Box 19		WORK UNIT NO.				
13a. TYPE OF REPORT MS Thesis		13b. TIME COVERED FROM TO		14. DATE OF REPORT (Yr., Mo., Day) 1985 December		
				15. PAGE COUNT 498		
16. SUPPLEMENTARY NOTATION						
17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)			
FIELD	GROUP	SUB. GR.	Local Area Networks, Network Interface, Protocols, UNID, ISO Reference Model, X.25, X.121, TCP/IP.			
09	05					
19. ABSTRACT (Continue on reverse if necessary and identify by block number)						
Title: DEVELOPMENT AND IMPLEMENTATION OF THE X.25 PROTOCOL FOR THE UNIVERSAL NETWORK INTERFACE DEVICE (UNID) II						
Thesis Advisor: Dr. Gary B. Lamont						
Author: Mark W. Weber, BS EE, Capt USAF						
Approved for Public Release: IAW AFR 180-17 LTJN E. WOLAVER 16JAN86 Dean for Research and Professional Development Air Force Institute of Technology (AFIT) Wright-Patterson AFB OH 45433						
20. DISTRIBUTION/AVAILABILITY OF ABSTRACT CLASSIFIED/UNLIMITED <input checked="" type="checkbox"/> SAME AS RPT. <input type="checkbox"/> DTIC USERS <input type="checkbox"/>			21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED			
22a. NAME OF RESPONSIBLE INDIVIDUAL Dr. Gary Lamont		22b. TELEPHONE NUMBER (Include Area Code) 513-255-3576		22c. OFFICE SYMBOL AFIT/EN		

Block 19 (cont)

Abstract

This ~~research effort~~ describes the continued development of an improved Universal Network Interface Device (UNID II). The UNID II's architecture was based on a preliminary design project at the Air Force Institute of Technology. The UNID II contains two main hardware modules; a local module for the network layer software and a network module for the data link layer software and physical layer interface. Each module is an independent single board computer (SBC) residing on an Intel multibus chassis, complete with its own memory (EPROM and RAM), serial link interfaces, and multibus interface. The local module is an iSBC 544 and the network module is an iSBC 88/45. The network layer software supports the CCITT X.25, datagram option, protocol and the data link layer software supports the CCITT X.25 LAPB (HDLC) protocol. This report documents the further implementation of the CCITT X.25 protocol in the UNID II design. *for use*

END

FILMED

4-86

DTIC